EXTERNAL VERIFICATION OF SCADA SYSTEM EMBEDDED CONTROLLER FIRMWARE

THESIS

Lucille R. McMinn, Second Lieutenant, USAF

AFIT/GCS/ENG/12-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/12-02

EXTERNAL VERIFICATION OF SCADA SYSTEM EMBEDDED CONTROLLER FIRMWARE

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Insitute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Lucille R. McMinn, B.S.C.S.

Second Lieutenant, USAF

March 2012

AFIT/GCS/ENG/12-02

EXTERNAL VERIFICATION OF SCADA SYSTEM EMBEDDED CONTROLLER FIRMWARE

Lucille R. McMinn, B.S.C.S.
Second Lieutenant, USAF

Approved:

[illegible]	5 Mar 12
Major Jonathan W. Butts, PhD (Chairman)	Date
[illegible]	5 MAR 2012
Juan Lopez Jr. (Committee Member)	Date
Barry Mullins	5 Mar 12
Barry E. Mullins, PhD (Committee Member)	Date

Abstract

Critical infrastructures such as oil and gas pipelines, the electric power grid, and railways, rely on the proper operation of supervisory control and data acquisition (SCADA) systems. Current SCADA systems, however, do not have sufficient tailored electronic security solutions. Solutions available are developed primarily for information technology systems. Indeed the toolkit for SCADA incident prevention and response is unavailing as the operating parameters associated with SCADA systems are different from information technology systems. The unique environment necessitates tailored solutions. Consider the programmable logic controllers (PLCs) that directly connect to end physical systems for control and monitoring of operating parameters – compromise of a PLC could result in devastating physical consequences. Yet PLCs remain particularly vulnerable due to a lack of firmware auditing capabilities.

This research presents a tool developed specifically for the SCADA environment to verify PLC firmware. The tool captures serial data during firmware uploads and then verifies against a known good firmware executable binary. Attempts to inject modified and/or malicious firmware are identified by the tool. Additionally, the tool can replay and analyze captured data by emulating a PLC during firmware upload. The emulation capability enables verification of the firmware upload from an interface computer without requiring modifications to or interactions with the operational SCADA system. The ability to isolate the tool from production systems and verify the validity of firmware makes the tool a viable application for SCADA incident response teams and security engineers.

Dedicated to my mother, who taught me the value of science, logic, and critical thinking.

Acknowledgments

Many thanks to the Department of Homeland Security and ICS-CERT for sponsorship of this research. Major Butts – thank you for being an inspirational educator and officer, and providing me with incredible opportunities. To my family – thank you for unyielding support and motivation.

Lucille R. McMinn

Table of Contents

Page

Abstract . . . iv

Dedication . . . v

Acknowledgments . . . vi

List of Figures . . . x

List of Abbreviations . . . xiii

1 Introduction . . . 1
- 1.1 Background . . . 1
- 1.2 Motivation . . . 2
- 1.3 Research Purpose and Goals . . . 2
- 1.4 Approach . . . 3
- 1.5 Research Contributions . . . 4
- 1.6 Assumptions and Limitations . . . 4
- 1.7 Organization . . . 4

2 Background . . . 5
- 2.1 SCADA System Overview . . . 5
- 2.2 Current SCADA Security Landscape . . . 7
- 2.3 Security Research . . . 9
  - 2.3.1 Network Layer Security . . . 9
  - 2.3.2 Software Data Integrity . . . 11
- 2.4 Embedded Device Security . . . 11
  - 2.4.1 Hardware Level Attestation Implementations . . . 12
  - 2.4.2 Software Level Attestation Implementations . . . 13
  - 2.4.3 Embedded Device Configuration Management . . . 15
- 2.5 Integrated Circuit Supply Chain Management . . . 16
- 2.6 Industrial Control System Security Recommendations . . . 18
- 2.7 Advanced Threat on SCADA Systems . . . 19
  - 2.7.1 The Evolving Threat Trend . . . 20
- 2.8 Background Summary . . . 21

3 Methodology . . . 22
- 3.1 Problem Definition . . . 22
  - 3.1.1 Goals and Hypothesis . . . 22

3.1.2 Approach . . . 22
3.1.3 Baseline Analysis and Emulation . . . 23
3.1.3.1 Baseline Capture and Analysis . . . 23
3.1.3.2 Emulation . . . 24
3.1.4 Firmware Verification . . . 24
3.1.5 Assumptions and Limitations . . . 25
3.2 Environment . . . 27
3.3 Evaluation Technique . . . 28
3.3.1 Baseline Capture and Analysis Test Cases . . . 28
3.3.2 Emulation Test Cases . . . 29
3.3.3 Firmware Verification Routine Test Cases . . . 30
3.3.3.1 Control Test Cases . . . 30
3.3.3.2 Modification Test Cases . . . 31
3.3.4 Evaluation of Firmware Modification . . . 32
3.4 Methodology Summary . . . 32

4 Analysis and Results . . . 33
4.1 Tool Development . . . 33
4.1.1 Protocol Patterns . . . 33
4.1.1.1 Identifying Protocol Fields and Patterns . . . 33
4.1.2 Data Emulation . . . 35
4.1.3 Firmware Verification . . . 37
4.2 Results . . . 38
4.2.1 Baseline Capture and Analysis Results and Emulation Results . . . 38
4.2.2 Firmware Modification . . . 39
4.3 Analysis . . . 39
4.3.1 MD5 Hash, Bitwise Check, and Tool Verification Routines . . . 40
4.4 Discussion on Limitations . . . 40
4.4.1 Platform Limitations . . . 40
4.4.2 Development Limitations . . . 41
4.5 Analysis Summary . . . 42

5 Conclusions and Future Work . . . 43
5.1 Conclusions . . . 43
5.2 Impact . . . 44
5.2.1 Impact to Manufacturers . . . 45
5.2.2 Impact for Industry . . . 46
5.2.3 Impact for Security Professionals . . . 46
5.2.4 Potential Implementations . . . 47
5.2.5 Potential Weaknesses . . . 47
5.3 Future Work . . . 47
5.3.1 Increased Testing . . . 47
5.3.2 Improved System Compatibility . . . 48

5.3.3 Increased Tool Security 48
5.3.4 Potential to Test All Possible PLCs 48
5.3.5 Traffic Generation 48
5.3.6 Alternative PLC Layer Security 49
5.4 Concluding Remarks 49

Appendix A: Typical ControlFLASH Firmware Load Process 51

Appendix B: Pseudocode 56

Appendix C: Tool Class Diagrams 62

Appendix D: Tool Performing Test Cases 68

Appendix E: Test Case Modifications 73

Bibliography 79

List of Figures

Figure Page

2.1 A generic SCADA system setup 6

2.2 PLC software layers 7

2.3 Example non-traditional SCADA system inputs 9

3.1 Passive capture and baseline analysis setup. 23

3.2 Emulator setup. 25

4.1 Example packet fields. 34

4.2 Protocol and firmware bytes in a version 15 capture. 36

4.3 Subset of blocks created by the parsing and analysis process of a version 15 PLC data capture. 37

4.4 Example firmware capture verification cases. 38

4.5 Firmware verification routine results. 39

5.1 Device centric security. 45

A.1 Open ControlFLASH. 51

A.2 Select PLC type. 52

A.3 Select PLC from device network. 52

A.4 Enter PLC designator number. 53

A.5 Select firmware revision number. 53

A.6 Begin upload. 54

A.7 Uploading in progress. 54

A.8 Power cycle upon upload. 55

A.9 Successful Install. 55

B.1 Data Capture Algorithm 56

B.2 Protocol Analysis Algorithm 57

B.3 Protocol Analysis Block Grouping Algorithm 57

B.4 Protocol Analysis Mismatch Resolving Algorithm 58
B.5 Protocol Analysis Escape Byte Check Algorithm 58
B.6 PLC Emulator Algorithm 59
B.7 PLC Emulator Packet Production Algorithm 59
B.8 Firmware Verification Algorithm 60
B.9 Firmware Verification Mismatch Algorithm 61
C.1 Main Class and Helper Classes. 62
C.2 Passive Serial Capture Classes. 63
C.3 Serial Capture Analysis and Profile Creation Classes. 64
C.4 PLC Emulator Classes. 65
C.5 Captured Firmware Verification Classes. 66
C.6 Firmware Parse and Check Classes. 67
D.1 Tool function selection screen. 68
D.2 Version 15 capture verification test. 68
D.3 Version 16 capture verification test. 69
D.4 Byte added to firmware verification test. 69
D.5 Byte removed from firmware verification test. 70
D.6 Byte added to protocol data verification test. 70
D.7 Byte removed from protocol data verification test. 71
D.8 Byte modified in firmware verification test. 71
D.9 Byte modified in protocol data verification test. 72
E.1 Version 15 firmware screenshot. 73
E.2 Version 15 firmware screenshot. 74
E.3 Version 15 capture screenshot. 74
E.4 Version 16 capture screenshot. 75
E.5 Byte added to firmware data. 75

E.6 Byte removed from firmware data. 76

E.7 Byte added to protocol data. 76

E.8 Byte removed from protocol data. 77

E.9 Byte modified in firmware data. 77

E.10 Byte modified in protocol data. 78

List of Abbreviations

Abbreviation		Page
SCADA	supervisory control and data acquisition	1
USB	universal serial bus	1
PLC	programmable logic controller	2
HMI	human machine interface	5
RTU	remote terminal unit	5
ROM	read only memory	5
BIOS	basic input/output system	5
RAM	random access memory	6
I/O	input/output	7
IT	information technology	11
FPGA	field programmable gate array	12
SWATT	software-based attestation for embedded devices	13
OMAP	one-way memory attestation protocol for smart meters	15
IEEE	Institute of Electrical and Electronics Engineers	15
IC	integrated circuit	16
DARPA	Defense Advanced Research Projects Agency	16
IRIS	Integrity and Reliability of Integrated Circuits	16
EEPROM	electrically erasable programmable read-only memory	17
DOD	Department of Defense	17
DHS	Department of Homeland Security	18
US-CERT	United States Computer Emergency Readiness Team	18
ICS-CERT	Industrial Control Systems Cyber Emergency Response Team	18
NIST	National Institute of Standards and Technology	18

ISA	International Society for Automation	18
DOE	Department of Energy	18
FERC	Federal Energy Regulatory Commission	18
NERC	North American Electric Reliability Corporation	19
DLL	dynamic linked library	20
BCC	block check character	35

External Verification of SCADA System Embedded Controller Firmware

1 Introduction

1.1 Background

The nation's critical infrastructure depends on secure, reliable supervisory control and data acquisition (SCADA) systems, which provide critical control, communication, and monitoring capabilities over geographically dispersed locations [1], [2]. As SCADA systems increasingly interconnect via unsecured networks, security solutions have focused on creating logical and physical boundaries between systems and the network layer [3], [4], [5]. Even with network isolation, however, additional attack ingress points have manifested in SCADA systems.

Consider, for example, the Stuxnet virus. Among the many implications associated with Stuxnet, the attack methodology demonstrated the ability to exploit critical systems through nontraditional inputs [6]. Indeed, the initial ingress point was not associated with the compromise of a standard information technology system via a network access point (e.g., compromise of firewall through the Internet). Instead, the virus propagated via a physical medium, eventually gaining access to an internal trusted node in the SCADA network. Although the attack vector by itself is not specifically unique - USB drives were placed by a foreign intelligence agency to exfiltrate sensitive data once inserted into U.S. Central Command systems [7] - it is the first documented case of a targeted attack against a specific SCADA system that resulted in kinetic effects [6].

SCADA systems are comprised of interconnected nodes consisting of master controlling systems, end devices, communication links and various support systems [8]. A cyber attack on a SCADA system assumes that vulnerabilities stem from the ability of an external actor to gain access to and communicate with nodes on the network. As a result, today's security perspective is to isolate and limit access points to the system from any external network [5]. While it is true that the network vulnerabilities currently considered are valid input vectors for an attack, they are not the only vectors that introduce untrusted and unvalidated inputs into a trusted system.

1.2 Motivation

The current tool set for attack response and mitigation is inadequately tailored to SCADA systems [9], [10]. Generally, the available resources are modified network-based tools adapted to incorporate the SCADA environment (e.g., packet capture tools, general operating system analysis and network-based intrusion detection systems). Although these tools provide a level of protection and analysis in a broad sense, tailored security solutions are needed to address the emerging threat specific to SCADA systems. Perhaps the most pressing concern is verifying the proper operation of field devices, such as programmable logic controllers (PLCs), which directly control and monitor the end physical systems. These devices typically operate 'below' the network layer and have few security mechanisms. As demonstrated by Stuxnet, unimpeded manipulation of these devices can have direct physical consequences. Indeed, there are currently no viable tools readily available to validate the proper operating parameters of PLC devices controlling the nation's critical infrastructure.

1.3 Research Purpose and Goals

This research examines a method for validating PLC firmware. Firmware, in the most basic sense, is fixed microcode that provides the bridge between hardware and higher-level programmable software on a device. An attacker that can gain access to and manipulate firmware has full control over the functionality of the device and can mask actions from detection.

The goal of this research is to develop a tool that validates PLC firmware and ensures that any attempt to alter the firmware is detected. The tool is intended to capture a PLC firmware load and identify any modifications from an established baseline to include data insertion, deletion, and modification. The tool should not introduce additional input vectors into the SCADA system. Additionally, an ideal implementation enables assessment with no impact to the operational SCADA system.

1.4 Approach

The tool is designed for an environment consisting of an Allen-Bradley FlexLogix 5434 PLC and Rockwell Software's RSLogix 5000 interface tool. The tool captures and audits serial communication using the Allen-Bradley DF1 full duplex protocol set.

Once captured, the firmware is validated against a known good baseline using a comparative analysis derived from observed firmware load characteristics. In the most basic form, identifying modified firmware is best accomplished by comparing against a known good firmware baseline. Note that simple hash comparison for firmware is not as straightforward as checking for modified files in a traditional operating system. Indeed, the requirement to capture and analyze the data during load, coupled with the comparison, makes firmware validation nontrivial.

Note that no two firmware loads are precisely the same due to changing protocol fields. Comparative analysis of captured data enables the tool to identify and account for these differences while validating the firmware bytes are consistent with the baseline. The altered firmware loads are evaluated using the tool's firmware verification routine, MD5, and bitwise check routines to demonstrate the ineffectiveness of MD5 and bitwise routines at identifying modifications.

1.5 Research Contributions

This research serves to introduce a tool to detect the loading of malicious or altered firmware to a PLC device. Application of the tool can assist incident response teams in pinpointing malware access points, as well as security engineers in providing assurance in the validity of PLC operations.

1.6 Assumptions and Limitations

Firmware verification is valid under the assumption that firmware modifications are manifested during a firmware loading session. However, it may be possible for malware to infect other components of a SCADA system. The tool requires a known good baseline to successfully validate firmware captures. This research assumes a known good baseline for comparison. Additionally, because tool evaluation is in a controlled environment, it is assumed that test cases are indicative of similar classes of firmware modifications (e.g., a one byte length increase is indicative of an *n* byte increase). Finally the variety of PLC manufacturers, configurations, and capabilities is a limiting factor for developing a broad spectrum tool. It is expected that further functionality can be added in future development iterations.

1.7 Organization

The remaining thesis is organized as follows. Chapter 2 provides background information, pertinent research, and literature review. Chapter 3 presents the research methodology. Chapter 4 presents results and analysis. Chapter 5 discusses research conclusions, impact, and future work.

2 Background

2.1 SCADA System Overview

Critical infrastructures such as oil and gas pipelines, the electric power grid, and railways, rely on the proper operation of SCADA systems. SCADA systems support various applications and may extend over thousands of miles. Operators remotely control and monitor SCADA systems from centralized monitoring stations, typically through a human machine interface (HMI). Field devices, such as are either PLCs or remote terminal units (RTUs), automate physical system operation by implementing digital control messages into actions (e.g., opening or closing breakers and valves, monitoring alarm conditions, and collecting system and environment data from sensors).

Remote field devices communicate with master control devices though a hierarchical communication paradigm. Master devices send direct messages requesting data or specifying actions to remote field devices, which reply with generated response messages. The master may be notified if the device detects an alarm condition. Figure 2.1 illustrates a simple control system consisting of a master device and a field device. The master device can send commands to control pumps or valves, or request sensor meter values. The field device replies with appropriate responses.

A PLC contains a microprocessor and read-only memory (ROM) or flash memory for storing firmware and control logic [5],[8]. A PLC typically has three software levels as shown in Figure 2.2. Modifiable layers include the control program, the firmware, and in some instances, the basic input/output system (BIOS). At the lowest level, the BIOS or other hard coded firmware occupies a portion of read-only memory and provides basic functionality. This routine executes when the PLC is powered on, initializing the PLC state and loading the operating system [11], [12]. If the operating system, or firmware, has not been installed and needs to be loaded, the BIOS handles this as well. Note that in

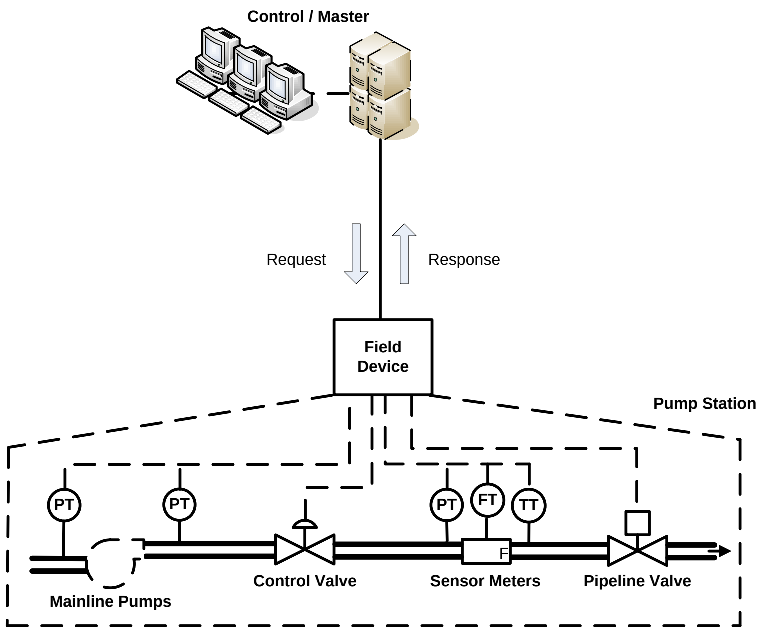


Figure 2.1: A generic SCADA system setup

many PLCs, the BIOS is not electronically modifiable. However in instances such as the Siemens' S7 modular embedded controllers which run embedded Windows XP, the BIOS is reprogrammable [13].

The operating system, referred to as firmware, runs on top of the BIOS layer. Firmware can range from simple proprietary software to high level processor demanding software such as embedded Linux or Windows versions. The loaded firmware has the ability to control random access memory (RAM), the runtime environment, sensors and actuators, and serves as the interpreter for the user defined logic program meant to control the SCADA system. Note that Allen-Bradley uses in-house created firmware for their PLCs [14].

The control program represents the highest externally electronically modifiable level and is typically a ladder logic or high level graphical or textual logic program [15]. In the

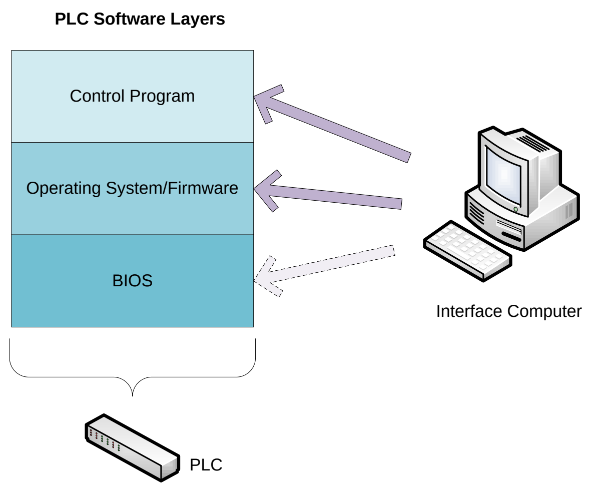


Figure 2.2: PLC software layers

case of the input/output (I/O) modules - built in I/O capabilities are not usually reprogrammable, however some devices do allow I/O module updates to enable PLC communication using new protocols. For scope simplification, this research focuses only on the general PLC level and not any components that may have specific vulnerabilities.

Note that the terms PLC and RTU are often interchanged. Although similar in functionality, RTUs are less advanced in terms of independent processing capability and rely on remote monitoring and control [8]. This research focuses on PLCs that have modifiable firmware and logic program layers.

2.2 Current SCADA Security Landscape

Current SCADA security focuses on traditional network security constructs. Firewalls and intrusion detection or prevention systems are used to create a defense-in-depth security architecture, and many SCADA system specific developments focus on encryption for information security through added confidentiality [16], [17]. However, encrypted data also increases the difficulty of auditing communication data [8].

Additionally, SCADA security developments often require modifying or adding system components, which can hinder real time performance [18], [19], [20], [21].

While network defense is critical to security, network based attacks may not be the primary method of exploitation. Stuxnet infiltrated a non-networked environment via a nontraditional vector – a USB [22]. Indeed, critical infrastructure is a high value target; an advanced persistent threat will find an input vector to even the most secure system [23], [24], [25]. Considering the various non-networked input vectors, security must be applied beyond the network layer [15].

PLCs are typically monitored and interfaced via a remote human machine interface [11]. The PLC controls SCADA system equipment and reports information about the site conditions to the remote monitoring station. A PLC under the control of a malicious user can produce devastating effects, as evidenced recently by Stuxnet and various historical attacks [6] [26].

The PLC presents three main vectors for attack: hardware, firmware and programming. Hardware security requires a trusted supply chain or methods to thoroughly test the acquired end device. Hardware is the lowest layer of abstraction and, at some level, must be trusted. Programming modifications alter PLC functionality and can be manipulated easily with compromise of or access to the specific PLC management software. Manipulation of PLC programming, however, is readily identifiable as PLC programs are interpreted instead of compiled and cannot be masked from external PLC inspection. When considering modifications to the PLC, firmware modification is the most intrusive and least detectable vector; there are no readily available methods to easily extract the firmware once loaded on a PLC [27], [28]. Exasperating the problem, as demonstrated in Figure 2.3, are the number of potential input vectors that enable firmware alteration (e.g., programming computers, SCADA control systems and access to the

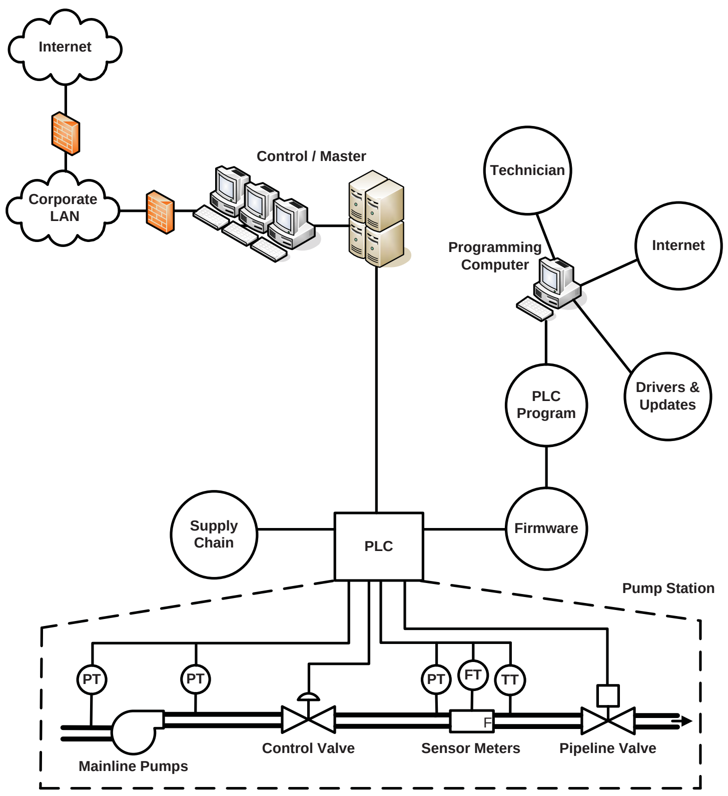


Figure 2.3: Example non-traditional SCADA system inputs

firmware update software). Indeed, any access point to the PLC or access to the firmware code to be uploaded provides an avenue to alter the end device's firmware.

2.3 Security Research

2.3.1 Network Layer Security. To validate the need for increased protocol security, a series of possible attacks against the MODBUS serial and TCP protocols as well as against the DNP3 protocol were identified by Huitsing *et al.* and East *et al.* [29], [30]. The identified effects range from intermittent disruptions to loss of awareness and control of the system. They analyzed attacks from the standpoint of interception, interruption, modification, and fabrication attack instances against the master station, end devices, or the communication network.

The majority of security developments redesign SCADA networks to add a layer of encryption. Fovino *et al.* suggests modifying SCADA architecture to make communication adhere to a signature. Additionally, they describe an architecture that has multiple validation 'filtering units' for each of the end devices such that a mathematical majority of the units per device must be corrupted to send a malicious packet [31]. The implementation mitigates unauthorized commands, man-in-the-middle attacks, replay attacks, and malicious packet attacks. However, this architecture must incorporate additional cryptography supporting devices. Another secure cryptographic-based environment is described by Pal *et al.* The authors note that the limited computation capacity, memory capacity, bandwidth, and real-time demands of a SCADA network environment pose significant technical challenges when implementing cryptography because key storage, encryption, and decryption require non-trivial processing time. They propose multiple architectures, each with individual advantages, of key storage and distribution among a network by a master unit [17].

Intrusion detection systems are proposed by both Morris and Pavurapu, and Wei Gao *et al.* [21], [25]. Morris and Pavurapu propose a bump-in-the-wire style device retrofitted into a network that monitors and handles encryption, analysis, and logging of all packets. This device would prevent response injection, command injection, and denial of service attacks. Wei Gao *et al.* alternatively researched adding network security without the complications of encryption [25]. Their research provides specific details on an intrusion detection system developed from a neural network algorithm with statistics on the ability to correctly detect malicious traffic. This research focuses on developing a suitable intrusion detection system, and not on adding a layer of encryption.

Research by Mander *et al.* and Gilchrist examines building security into the DNP3 protocol [16], [32]. Mander *et al.* proposes adding object based security rules based on packet fields such as function code and data field values to prevent modifications to end device configuration settings; non-conforming packets are dropped. Gilchrist presents adding a secure authentication standard to the DNP3 protocol using keyed-hash message authentication codes. The development was implemented by the DNP3 IEEE 1815 standard [33]. The specification applies only to how the authentication is negotiated, and not how keys are stored or distributed – the standard leaves encryption details to the user's discretion. The authors imply that encryption is intended to be used with the standard, though the protocol standards do not require encryption for functionality.

Adding layers of network security to SCADA networks proactively mitigates the threat of external unauthorized access; however, it is important to note that these methods only protect the perimeter of a network. End devices such as PLCs typically are not capable of deploying their own information technology (IT) protection methods [8].

2.3.2 Software Data Integrity. Software level data integrity checking is pertinent to analyzing PLC software integrity. Data integrity checks are performed through a variety of hashing functions [34]. MD5 is a commonly used hash function to check if a file has been modified [35]. A known good MD5 is compared against the MD5 of the data in question. Equivalent results provide a measure of integrity that the data remains unmodified.

2.4 Embedded Device Security

Current SCADA systems primarily use commercially available operating systems for the HMI system. Vulnerabilities in these systems are openly attainable, allowing an attacker to gain expertise on a platform without internal access to SCADA system components [36]. PLCs however, are more widespread with each manufacturer and model typically implementing different firmware. Because PLCs are in effect a microprocessor device, an analysis of the current research on embedded devices is important to garner insight to applicable security solutions.

Embedded device security focuses on controlling the onboard memory of a device and attesting to its validity. This is applicable to a typical PLC design because the PLC firmware and logic programs are stored in ROM or flash memory and RAM respectively. Assuring that a device's static configuration or its dynamic execution state is secure increases confidence that a device has not been compromised [18]. Current research on embedded device security involves attestation that the device state has not been changed. Research has been done on secure protocol development, software level attestation, hardware level attestation, and a combination of these designs. Attestation can be accomplished by integrating a hardware device as an external verifier, or on the software level where a trusted portion of code exists to check the state of the device. Additionally, research has been accomplished for remote schemas to allow verification without physical presence over a network. Remote attestation requires a secure communication protocol. A discussion of embedded device security research follows.

2.4.1 Hardware Level Attestation Implementations. Implementations from research executed by Basile *et al.*, Feller *et al.*, and Khan *et al.* add hardware level components to externally verify the device state [18], [37], [38].

Research by Basile *et al.* focuses on detecting if executed code has been modified by utilizing an field programmable gate array (FPGA) to build a secure architecture. The authors' goal is to engineer a setup that makes it difficult to create a successful real world attack. Various network and environmental attack scenarios are addressed. However, the authors do note that this method of protection is not intended for high value targets - a motivated attacker with resources could still compromise an FPGA.

TinyTPM by Feller *et al.* was presented to port trusted computing to FPGA devices by adding a validation module that consumed few resources in static logic [37]. TinyTPM performs bootstrapping functions in order to verify the system state before any dynamic logic execution can occur. The module uses cryptography but consumes fewer resources than predecessors' Trusted Platform Module implementations in order to add security. Khan *et al.* presents another verification method for embedded systems utilizing an external FPGA to store a hash and to check memory on device reset [38]. If the computed hash matches the stored hash during the boot process, then the power up continues to allow the device to run, if they do not then the device stays in reset mode. This method adds some delay at boot time; however, it does not affect the system's functionality after verification occurs. This could be beneficial for PLC security; however, changes would be needed to implement boot time checking in a critical process environment as a no-start situation is unacceptable and may create a denial of service situation.

2.4.2 Software Level Attestation Implementations. Adding hardware devices is not always feasible, and can be expensive. Software based attestation seeks to discover that an environment is safe by using provably secure methods.

Software-based attestation for embedded devices (SWATT) is one of the earliest projects that presents the idea of using a purely software based external verifier [39]. SWATT acts as in intermediate between the device and external actors communicating with the device. A challenge-response protocol is used to only allow validated devices to communicate. The technique is similar to secure bootstrapping; however, SWATT is unique because it does not require additional hardware to externally verify the system was in a secure state. Note that complete knowledge of the hardware is required for SWATT to be implemented successfully. The verification process is alleged secure because any changes to the checking routine would cause a measurable time delay. However, one of the flaws with SWATT is the assumption that the device is in a trusted environment. The authors note that because of the untrusted environment, any integrity checking functions stored on the unit could be modified by an attacker, making them insecure as well.

Igure *et al.* present modern improvements on general software based attestation [40]. Though not specific to embedded devices, the paper discusses ensuring the system is in a secure state by allowing only the attestation process to be active, and then by scanning memory to verify this assumption. The verification process relies on the idea that computation has access to all free RAM, and if less than the correct amount is free, a noticeable delay will occur, indicating the presence of malware. When executing the validation routine, all processes should be swapped out, including the kernel, to maximize free RAM. The validation is then run by a monolith kernel. If less than the anticipated amount of RAM is free, then secondary memory must be used. The security of this method is guaranteed as long as the assumption holds that RAM is faster to access than secondary memory, which is a more secure method than just pure software validation alone.

A more thorough but computationally demanding approach is taken by AbuHmed *et al.* The authors claim that attesting a devices entire memory space is the most secure as it does not allow the attacker to save the original memory in a different location [41]. Any additional memory can be filled with incompressible noise so that an attacker cannot compress data in unused memory to free memory for malicious code additions.

Providing firmware updates remotely, or over the air, is another problem in memory attestation. Updating every PLC directly is infeasible in systems with abundant or difficult to reach controllers. Nilsson *et al.* presents a security framework for secure firmware updates by connecting to a trusted portal and downloading updates using a secure protocol [42]. The firmware is verified by using hardware virtualization and running two systems - one system is the embedded controller system and the other is the verification system. This allows self verification of updates remotely. The secure protocol is a unique development to increase secure communication. Instead of using a trusted remote verifier as with SWATT, Nilsson *et al.* seeks to use virtualization to separate a trusted microkernel and an untrusted embedded operating system. The downloaded binary can be verified before it is flashed and the secure protocol protects against third party modifications from the trusted sender to the receiver. This schema can verify correct downloads and successful flashing for firmware updates, however, it is still an incomplete solution to PLC firmware security because it does not account for modified firmware sent from a trusted source.

Research for secure remote memory attestation was accomplished by Kyungsub *et al.* [20]. They designed a one-way memory attestation protocol for smart meters (OMAP). OMAP utilizes a random memory traversal checksum to protect against local attacks and forwards the checksum to a utility to verify if the checksum corresponds to unmodified memory. Modifications are detected through a random memory traversal checksum computation. This implementation relies on the memory verification routine to be secure. The research by Ce Meng *et al.* notes with heterogeneous components in a system, the overall trust of a system depends on trusting its components [43]. They discuss attesting to the building process in addition to remote attestation methods.

The limitation with purely software based attestation is that depending on the knowledge of the attacker, the firmware in an embedded device may be completely reprogrammed yet imitate a good system. The validation methods add time and processing overhead, which may degrade real time constrained systems or may be infeasible for systems with limited memory or processing capability. Hardware security solutions require system compatibility and can limit device functionality. Currently operating critical infrastructure systems would require retroactive installation to include hardware modifications. While attestations on both levels afford improvements for general embedded devices, neither is tailored to provide the high level of verification necessary for critical infrastructure systems.

2.4.3 Embedded Device Configuration Management. Applying security retroactively is inadequate for a mature security posture. In addition to adding device security, controlling device updates and modifications are imperative for security maintenance. The Institute of Electrical and Electronics Engineers (IEEE) published a best practices guide for firmware control on microprocessors [44]. IEEE suggests that microprocessors should have strict configuration management. Every update should be tested before it is implemented to ensure that the current functionality will not be impacted, and the previous version should always be backed up in case of malfunction. Firmware should only be updated if it is required for functionality. However, the guide does not discuss vectors that malware might use to compromise firmware or an embedded device, or how to mitigate threats.

2.5 Integrated Circuit Supply Chain Management

Systems that utilize integrated circuit (IC) components must make the assumption that the hardware device operates as intended. Regardless of how robust a security system is, if it is running on insecure hardware, then its trustworthiness is undermined. Even with extensive testing, it is infeasible to check every layer on every chip for correct functionality. Thus, trusting an IC extends trust to every point along the supply chain. Counterfeit parts are an industry issue, highlighting the feasibility for compromised parts to be integrated into a system [28]. Legacy systems are particularly susceptible to compromise as replacement parts are often obsolete.

Supply chain management is critical to mitigating risks in embedded devices before they reach the end system. Since all technology relies on trustworthy hardware, significant research has been done, as well as government programs put in place, to efficiently detect and prevent hardware modification. The Defense Advanced Research Projects Agency (DARPA) conducted the “TRUST in IC’s” Effort in 2007, a three year initiative to develop integrated circuits compromise detection methods [45], [46]. DARPA continued this research in 2010 with the Integrity and Reliability of Integrated Circuits (IRIS) initiative to determine if an IC has been modified in a malicious manner [27].

Supply chain attacks can be grouped into three general categories - circuitry modification, programmable hardware attack, or firmware attack [47]. Circuit modification deals with altering the physical components of an IC. An attacker in the supply chain can also reprogram hardware through electrically erasable programmable read-only memory (EEPROM).

Firmware, which is an abstraction above programmable hardware, can also be modified to change device functionality. Once a device has been modified, it is difficult to detect modifications in a nondestructive manner. Exploits can be injected during design, production, distribution, or maintenance phases of an IC's lifecycle. Design phase exploits are especially subversive because the malicious logic is designed into the chip without any need for physical interception and modification.

There exist documented cases of 'kill switches' built into microprocessors to disable functionality [48]. Analyzing IC layers to verify the absence of malicious logic is difficult and most methods destroy the IC in the process. The Department of Defense (DOD) has taken measures to increase trust in hardware by validating American commercial plants as trusted IC foundries, however this does not guarantee a trusted product. Additionally, faults can be engineered such that ICs operate normally, but then fail significantly before the predicted lifetime.

Hardware level changes introduce myriad attacks, including timed attacks and disabling encryption security. Furthermore, ICs can be modified even after they have been manufactured. While this requires significant resources and blueprint knowledge, the possibility is not unreasonable for a well funded and motivated attacker [48]. Solutions that do not involve IC deconstruction utilize side channel analysis. Side channel analysis operates under the assumption that inserting a trojan on the hardware level will degrade chip performance or otherwise alter functionality in an observable manner. Hardware trojan detection methods are not standardized and benchmarking has only been recently introduced. Even with detection methods, not all hardware can be thoroughly checked.

Commercial off-the-shelf systems and legacy systems continue to be a difficult problem, especially considering the global market for IC components [49].

Hardware threats undermine software security as software security can be bypassed by hardware. Integrated circuits are inherently complex. Malicious logic detection in hardware is difficult, especially for logic or time activated trojans which lay dormant until a logic condition is occurs [50].

2.6 Industrial Control System Security Recommendations

In 2009 the Department of Homeland Security (DHS) published the National Infrastructure Protection Plan [2]. The Government Accountability Office also published testimony on the cyber threat to critical infrastructure [9], [24]. The DHS established the United States Computer Emergency Readiness Team (US-CERT) to respond to cyber incidents, including those in industrial control systems (ICS-CERT).

ICS-CERT provides many standards and best practices for security industrial automation systems, as well as a cyber security evaluation tool to help increase organizations' cyber security posture [51]. ICS-CERT also publishes vulnerabilities found in industrial control system equipment. They work with other agencies, including the National Institute of Standards and Technology (NIST), International Society for Automation (ISA), Centre for the Protection of National Infrastructure, Department of Energy (DOE), the Federal Energy Regulatory Commission (FERC), and the national laboratories to serve as a centralized resource for industrial control cyber security policy.

ICS-CERT presents detailed instructions on implementing system security. Establishing a segmented network with firewalls and demilitarized zones, applying configuration management, updating systems, adding authentication, training personnel in cyber security, conducting risk assessment, securing wireless connections, using encryption, limiting remote connections, using intrusion detection and prevention, and adhering to security standards encompass the general areas ICS-CERT publishes documentation on, accessible on their website [10].

NIST published a guide for SCADA system security [5]. NIST recommends restricting physical and logical access to the network, protecting individual SCADA components, and implementing redundancy. Additional recommendations are implementing a defense in depth strategy that involves lifecycle security on the system, adding layered network protection, employing encryption where possible, testing before changing the system, and utilizing logging and monitoring to track system activity.

The North American Electric Reliability Corporation (NERC) published a series of documents addressing cyber security and reliability standards for critical infrastructure cyber security [52]. Their standards focus on creating a cyber perimeter with defense in depth measures such as access controls, authentication, encryption, firewalls, logging, and intrusion detection systems [53], [54]. The industry cyber security standards published by these organizations focus on network security and access control. While these recommendations increase industrial control system security, they may not provide comprehensive security. Considering the current threat trend, vectors still exist to allow malicious input into the system.

2.7 Advanced Threat on SCADA Systems

According to a DARPA report by Collins, the United States has to assume adversaries are nation states with motivation, talent, time, and opportunity to do significant harm to the United States [46]. Hostile actors may seek to inflict economic or physical damage by attacking critical infrastructure networks. Possible attacks on a SCADA system include denial of service, eavesdropping, man in the middle, or hijacking a system through various infection methods such as a worm, virus, or trojan.

Malware falls into four generalized areas: malware that consumes resources, malware that degrades a system based on a trigger condition, malware that allows remote access, and malware that exfiltrates sensitive data [8]. Resource consumption, particularly on a resource sensitive SCADA network, degrades communications and uses bandwidth that the system requires to function properly. Trojans or timed attack malware propagate to a computer and then execute certain malicious logic to harm that computer's functionality, degrading SCADA system performance as well. Remote access malware, such as a rootkit, hides malicious logic at the kernel level in a system and provides an attacker to exercise with a back door entry point. This allows an attacker complete control over the computer. Exfiltration provides an attacker the knowledge necessary to engineer a targeted attack [8].

2.7.1 The Evolving Threat Trend. The threat trend has become increasingly targeted over recent years. Initially, targeted attacks originated from disgruntled insiders; however, this is no longer the case. Recent attacks targeting U.S. Government such as the exfiltration of Joint Strike Fighter data in April 2009, and the keylogger on remotely piloted vehicle control stations at Creech Air Force Base in September 2011 demonstrate the evolution of malware against high value targets [23], [55].

Stuxnet is a compelling example of the insufficiency of critical infrastructure security. Discovered in June 2010, Stuxnet infected a system not connected to the public network via a USB exploit. The virus analyzed the computer, only executing its payload if the system met various requirements. Stuxnet required a target running the Siemens' SIMATIC STEP 7 software with specific PLC CPU types and with specific numeric values present in the system data blocks. Once in place on the target system, Stuxnet modified the dynamic linked library (DLL) which controlled communication from the STEP 7 software to the PLC.

The PLC program was modified to include a malicious function block, and the modified DLL controlled the program's function block locate, read, and write capabilities.

The DLL modification could modify sent or received data, hiding the additional malicious function block from system operators [6], [22].

According to Symantec, developers of Stuxnet had extensive prerequisite knowledge of the target system [6]. However, the lack of overall system protection, input validation, and secondary fail safes enabled Stuxnet to infect approximately 100,000 hosts, with over 60% of hosts located in Iran, which resulted in physical damage. The future threat will likely capitalize on Stuxnet's innovations by both targeting specific systems and by using extensive intelligence to develop malware to subversively achieve a kinetic goal. Network security aimed to provide defense against hackers and general malware will not provide sufficient protection against attacks similar to Stuxnet.

2.8 Background Summary

Recent SCADA security research aims to mitigate cyber vulnerabilities through system architecture modifications or additions. Security researchers recommend adding encryption or attestation devices to protect memory from potential modifications. Standards and best practices recommend implementing traditional IT security solutions. Modifying field devices or adding devices may be detrimental to system performance and may not be feasible for all systems. Network security may not provide necessary protection against non-networked vectors similar to Stuxnet. Potential improvements upon current SCADA cyber security shortcomings are efforts that do not affect system operation and works with a system without modifications or additions to provide protection beyond the network layer.

3 Methodology

The ability to verify that a source device (i.e., interface computer) is sending unmodified firmware to a PLC requires three primary features: (i) the ability to capture communication data, (ii) the ability to analyze captured data and (iii) the ability to determine the validity of the firmware. This chapter describes the methodology for evaluating the effectiveness of the firmware validation tool for these three features.

3.1 Problem Definition

3.1.1 Goals and Hypothesis. Given a known good baseline, the firmware verification tool is expected to identify firmware modifications. This research postulates that MD5 hashing and bitwise check routines are insufficient because they do not consider stateful protocol communication fields. The three methods (i.e., MD5 hash, bitwise check, and the tool's verification routine) are examined by comparing a baseline firmware load against subsequent modified loads.

The primary goals of this research include: effectively capturing and analyzing firmware upload communications, effectively emulating a PLC to capture firmware loads independent of a PLC, and verifying firmware captures against known good baselines. Capturing, analyzing, and verifying firmware communication are the necessary functions for the firmware verification tool. The emulator functionality provides an implementation capability with operational feasibility.

3.1.2 Approach. Effective data capture and analysis is demonstrated by capturing and analyzing data using multiple firmware revisions. Each analysis produces a known good baseline. Emulating a PLC to independently capture firmware loads is demonstrated by mirroring each possible firmware version. Verifying firmware captures against known good baselines is demonstrated by executing the verification routine against representative firmware modification test cases. Analysis of the firmware verification tool's performance compared to the MD5 hash and bitwise check routines on representative test cases demonstrates the tool's relative ability to perform adequte firmware verification.

3.1.3 Baseline Analysis and Emulation.

Each firmware upload is executed in an identical manner. Keeping the loading routine constant for both the tool and the interface computer reduces the possibility for variations in the data transfer and allows the captured data to be representative of only the firmware loading process.

3.1.3.1 Baseline Capture and Analysis.

During the initial capture and baseline phase, a firmware load from the interface computer to the PLC is captured. Figure 3.1 illustrates the passive capture mode setup. The verification tool passively receives all transferred data. The captured data is then separated into data sent from the firmware interface computer and reply data sent from the PLC. Note that data from the interface computer contains the uploaded firmware bytes.

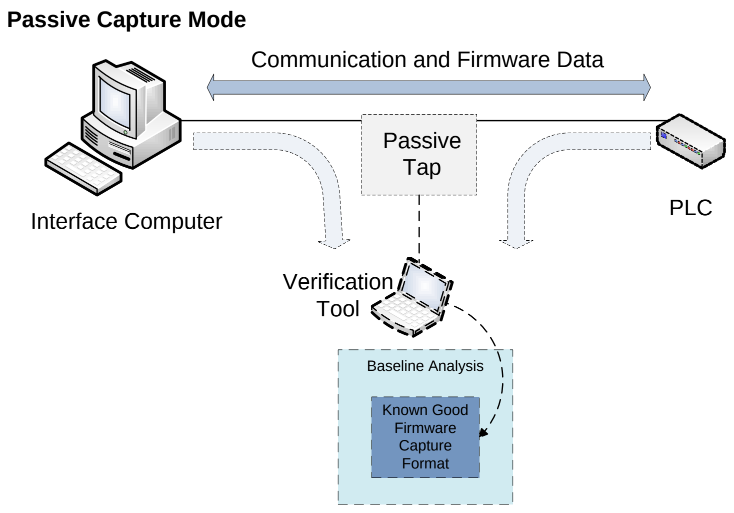


Figure 3.1: Passive capture and baseline analysis setup.

Baseline creation requires multiple captures. This allows the analysis routine to identify any stateful protocol packet fields. Stateful protocol packet fields contain elements that vary depending on data within the packet (e.g., checksum) or stateful variables (e.g., identification fields). The communication is parsed and variable bytes are evaluated against typical protocol field patterns [33], [56], [57], [58]. Once differences are accounted for, a protocol profile is created which contains the communication pattern. The pattern is then applied to received data to emulate future communication. The baseline is conducted for each firmware version. Baseline analysis is considered successful if the routine is able to create a complete baseline profile for captured data. The profile is considered complete if all stateful protocol packet fields are accounted for. If the baseline or profile success conditions are not met preventing successful profile creation, then the tool is unsuccessful.

3.1.3.2 Emulation. Given the baseline profiles, the PLC can be removed from the communication setup and the emulation environment can be implemented by directly connecting the interface computer to the verification tool. Figure 3.2 illustrates the emulator setup. The emulator connects to the interface computer and mimics PLC messages to initiate a firmware load. The emulator applies the baseline analysis data to the stored PLC communication data to create valid PLC reply packets. The emulation is conducted for each test case. The emulator is considered successful if the routine is able to utilize all available baselines to capture complete firmware loads. A firmware load is considered complete if the interface computer firmware loading program indicates the upload is complete.

3.1.4 Firmware Verification. Firmware modification tests are separated into two distinct types: changes to non stateful protocol data within a capture, and changes to firmware data within a capture. Note that stateful protocol fields are expected to change

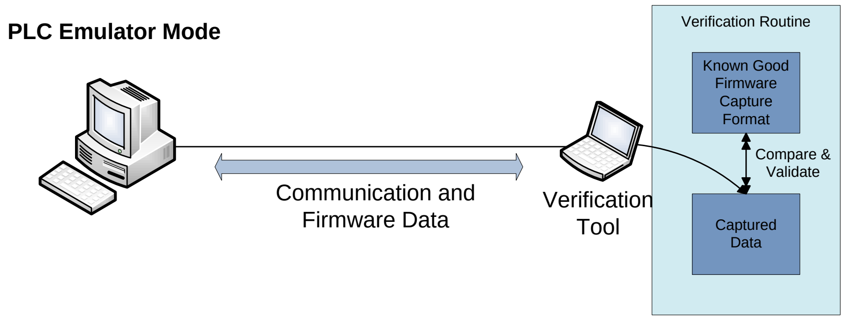


Figure 3.2: Emulator setup.

within captures, so there is not a class of stateful field modifications. Represented possibilities for data modification within a capture are data additions, subtractions, or content modifications. Unmodified data is also used as a control to indicate basic functionality. Test cases comprise a representative set of these possibilities for modification, as well as a control for each of the distinct types of modification. Each test modification and known good baseline is run against a standard MD5 hash function, a bitwise check function, and the tool's firmware verification function.

3.1.5 Assumptions and Limitations. Once the tool acquires a baseline, the emulation routine allows the to tool operate without the need for a PLC. As opposed to related research outlined in Chapter 2, the tool does not require modification of system architecture or system devices. It is assumed that the independent functionality, the lack of system modification, and the execution on a portable laptop makes the tool feasible for operational use.

The PLC and the interface computer are assumed to begin in a known good state. Additionally, when capturing, analyzing, and emulating data, it is assumed that the system is deterministic. Therefore multiple iterations of the same test are expected to produce identical outcomes.

The representative test cases are indicative of possible permutations of test cases. When checking modified firmware or protocol bytes, the location of the modification does not matter. The validation tool does not take location information into account; it analyzes only the differences between the baseline and firmware capture. Because the location of the modification does not matter, a location at any non stateful protocol field position or any firmware data field position is considered equivalent to any other modification of the same type. Therefore, a single modification of each class is sufficient to demonstrate other modifications of the class.

Stateful protocol fields are not tested by the validation routine test cases because they are assumed valid. The emulator routine testing encompasses the stateful protocol field testing because it is assumed that the interface computer is adhering to typical communication standards. This assumption is based on the fact that since the emulator only sends known packet PLC replies in order, the interface computer has the option of either replying with a valid successive request, or an invalid successive request. A valid request adheres to emulator expectations in terms of stateful protocol fields and therefore stateful fields remain valid. An invalid request may or may not deviate in terms of stateful protocol fields. Invalid data may either cause an error on the emulator side, which would indicate possible modification, or the next successive valid PLC reply will be sent, which would cause an error on the interface computer side, also indicating possible modification. If the case occurs that the interface computer sends data with incorrect stateful protocol fields but the emulator is able to send a valid reply and communication continues without an error on either side, then an error that is not identified occurs. An error such as this is caused by data corruption on the physical link layer, and not a purposeful modification. A system limitation is that errors in stateful fields are only accounted for if they cause errors or modifications to future packets.

It is possible some fields may change very slowly, and two successive firmware loads may not capture this behavior. It is assumed that allowing one load in between two successive firmware loads is sufficient to ensure all stateful protocol fields are manifested.

Consideration for changes with greater delays is discussed in areas for future work.

3.2 Environment

The evaluation environment consists of a standard Windows XP personal computer and an Allen Bradley FlexLogix 5434 PLC. The personal computer represents the interface computer designed to upload the RSLogix firmware. The interface computer has Rockwell Software's RSLogix 5000 suite installed as well as ControlFLASH 9.00.015, the firmware loading program.

For the initial baseline capture, the verification tool is connected to the primary communication line via a passive serial adapter tap, enabling interception of communication data while preserving communication between the interface computer and the PLC. For the subsequent emulator and verification phases, the tool is connected to the interface computer through a serial cable. The PLC and interface computer communicate using the DF1 protocol. The serial port is configured to correspond with the PLC's serial data capabilities, specifically a baud rate of 19200, 8 data bits, no parity, and 1 stop bit.

Firmware versions 15.06.01 or 16.21.12 are firmware load options for the FlexLogix 5434. Baseline firmware files and corresponding binary files are stored in the ControlFLASH program directory. Each individual ControlFLASH upload is executed with the same selection method to eliminate variations in the serial data transfer. Data capture starts when the ControlFLASH program is opened, and ends when the firmware upload shows complete on the interface computer.

3.3 Evaluation Technique

3.3.1 Baseline Capture and Analysis Test Cases. The two firmware versions 15.06.01 and 16.21.12 represent the possible versions that can be loaded onto the PLC. Each capture is run twice to identify stateful protocol fields. Two captures are the fewest number of captures necessary to identify changes between captures. Each test case produces a baseline communication profile.

The following steps outline the necessary actions for each test case. First the tool is placed in passive serial capture mode and the ControlFLASH environment is opened on the interface computer. The device is connected according to the passive setup as noted in section 3.1.3.1. The tool is configured for port specifications that match the specifications outlined in the environment section. Each serial line is chosen based on the corresponding device's tap and the data capture is started. Once capture has started, the 1794-L34 PLC is selected from the ControlFLASH catalog. Then the AB DF1-1, DF1 network is expanded and location 01, the FlexLogix L34 Processor is selected. Slot 0 is confirmed as the backplane position. The revision selected corresponds to the test case, and the update is started. Screenshots of the ControlFLASH firmware loading process are included in Appendix 5.4. The tool's data capture routine automatically ends once updating is complete and all data is received. The tool is then put in capture baseline and analysis mode. The saved capture data files created during passive capture are selected. The tool produces a corresponding baseline. Capture and analysis test cases consist of the possible firmware load options:

1. Version 15.06.01 to version 15.06.01 baseline capture followed by a second version 15.06.01 to version 15.06.01 baseline capture and an analysis of the two loads producing a corresponding baseline.

2. Version 15.06.01 to version 16.21.12 baseline capture followed by a second version 15.06.01 to version 16.21.12 baseline capture and an analysis of the two loads producing a corresponding baseline.

3. Version 16.21.12 to version 16.21.12 baseline capture followed by a second version 16.21.12 to version 16.21.12 baseline capture and an analysis of the two loads producing a corresponding baseline.

4. Version 16.21.12 to version 15.06.01 baseline capture followed by a second version 16.21.12 to version 15.06.01 baseline capture and an analysis of the two loads producing a corresponding baseline.

3.3.2 Emulation Test Cases. The four baselines created during baseline capture and analysis test case execution are used for the emulation test cases. This tests the tool's ability to emulate each loading option for updating the PLC.

The following steps outline the necessary actions for each test case. First the tool is placed in passive serial capture mode and the ControlFLASH environment is opened on the interface computer. The device is connected according to the emulator setup as outlined in section 3.1.3.2. The tool is configured for port specifications that match the specifications outlined in the environment section. Once emulation has started, the 1794-L34 PLC is selected from the ControlFLASH catalog. Then the AB DF1-1, DF1 network is expanded and location 01, the FlexLogix L34 Processor is selected. Slot 0 is confirmed as the backplane position. The revision selected corresponds to the test case, and the update is started. The tool's data emulation routine automatically ends once updating is complete and all data is received. Emulation test cases consist of the possible firmware load options:

1. Version 15.06.01 to version 15.06.01 emulation

2. Version 15.06.01 to version 16.21.12 emulation

3. Version 16.21.12 to version 16.21.12 emulation
4. Version 16.21.12 to version 15.06.01 emulation

3.3.3 Firmware Verification Routine Test Cases. The firmware verification routine test cases are run with each of the three verification routines: (i) MD5 hash, (ii) firmware bitwise check, and (iii) the tool's verification routine. The test case modifications are single byte modifications.

The following steps outline the necessary actions for each test case for the MD5 routine. The MD5 hash function is the built in MD5 checksum in the HxD program [59]. The test file and known good files are opened in the editor, and the MD5 checksum routine is selected from the analysis file option for each file. A resulting checksum is produced for each file. The test case checksum is compared to the known good checksum.

The following steps outline the necessary actions for test case for the firmware bitwise check routine. The firmware bitwise check is a built-in routine in the verification tool. The tool is placed in bitwise firmware check mode. The captured file path and the known good firmware file path are selected. The tool's output indicates if the firmware data is or is not contained within the capture.

The following steps outline the necessary actions for each test case for the tool's verification routine. The tool is placed in validate serial capture mode. The corresponding baseline file path and the test capture file path are selected. The tool's output indicates if the firmware is or is not equivalent.

3.3.3.1 Control Test Cases. Control test cases are unmodified firmware captures to test proper verification functionality. Baseline files used are respective baselines for the corresponding verification method. The MD5 hash verification and the tool's verification method use the delineated baselines whereas the bitwise check method uses the baseline firmware file binary.

1. No modification - version 15.06.01 to version 15.06.01 baseline versus a good 15.06.01 to 15.06.01 capture
2. No modification - version 15.06.01 to version 16.21.12 baseline versus a good 15.06.01 to 16.21.12 capture
3. No modification - version 16.21.12 to version 16.21.12 baseline versus a good 16.21.12 to 16.21.12 capture
4. No modification - version 16.21.12 to version 15.06.01 baseline versus a good 16.21.12 to 15.06.01 capture

3.3.3.2 Modification Test Cases. Modification test cases consist of single byte modifications to create a lengthwise change or a bitwise content change. Baseline files used are respective baselines for the corresponding verification method. The MD5 hash verification and the tool's verification method use the version 15.06.01 to version 15.06.01 baseline whereas the bitwise check method uses the 15.06.01 firmware file binary.

1. Lengthwise modification - baseline versus added single byte to the end of the firmware data embedded within a packet of the capture
2. Lengthwise modification - baseline versus subtracted single byte from the end of the firmware data embedded within a packet of the capture
3. Lengthwise modification - baseline versus added single byte to the end of the last captured data packet
4. Lengthwise modification - baseline versus subtracted single byte from the end of the last capture data packet
5. Bitwise modification - baseline versus modified single byte in the firmware data embedded within a packet of the capture

6. Bitwise modification - baseline versus modified single byte in the last captured data packet

3.3.4 Evaluation of Firmware Modification. Each test case outcome is either successful or unsuccessful at identifying modifications. The baseline capture and emulation tests consist of four test cases. The baseline is successful if the tool creates a baseline from captured data. The emulation is successful if the tool captures a firmware load.

Firmware modification test cases consist of four control and six modified cases executed by the three verification routines for a total of thirty outcomes. A successful outcome is the correct indication of the absence of modification for the control cases, and the presence of modification for the modified cases.

3.4 Methodology Summary

This chapter provides the methodology for evaluating the firmware verification tool. The data capture and analysis capabilities of the tool are examined to demonstrate the tool's ability to identify modified firmware and function independently of a PLC. The effectiveness is tested using a comparative analysis of the verification tool's routine to MD5 and bitwise check data integrity check routines. The test cases consist of representative modifications demonstrating possible modification classes.

4 Analysis and Results

4.1 Tool Development

A typical ControlFLASH firmware load involves selecting a PLC and corresponding available firmware version, and then proceeding through a firmware upload process. For each firmware file, the process first sends an update command, waits for the PLC to power cycle, sends the firmware data, and then waits for a success or failure response and a final power cycle. The tool baselines and emulates the firmware loading process. For the initial baseline phase, the tool captures data from the interface computer to the PLC. When the baseline has been captured, the tool runs an analysis on the captured communication and creates an emulation profile. Subsequently, the tool can connect directly to an interface computer to download and independently verify firmware. The tool was developed in C# using Visual Studio 2008 and executes on the Windows 7 64 bit operating system. The tool requires two serial ports or serial port adapters on the host computer to capture serial data.

4.1.1 Protocol Patterns. The verification tool requires adequate functionality to capture and interpret data from Allen-Bradley devices utilizing the DF1 full duplex protocol. Patterns are applied to a known good PLC capture, identifying fields to create a baseline profile.

4.1.1.1 Identifying Protocol Fields and Patterns. The tool employs a brute force optimization technique for identifying the protocol fields and patterns. First, start fields are used to group received data into packet structures, separating each block of data by start field. The start field identification algorithm begins with the first byte in the capture as the initial field and seeks to successively increase field length while maintaining

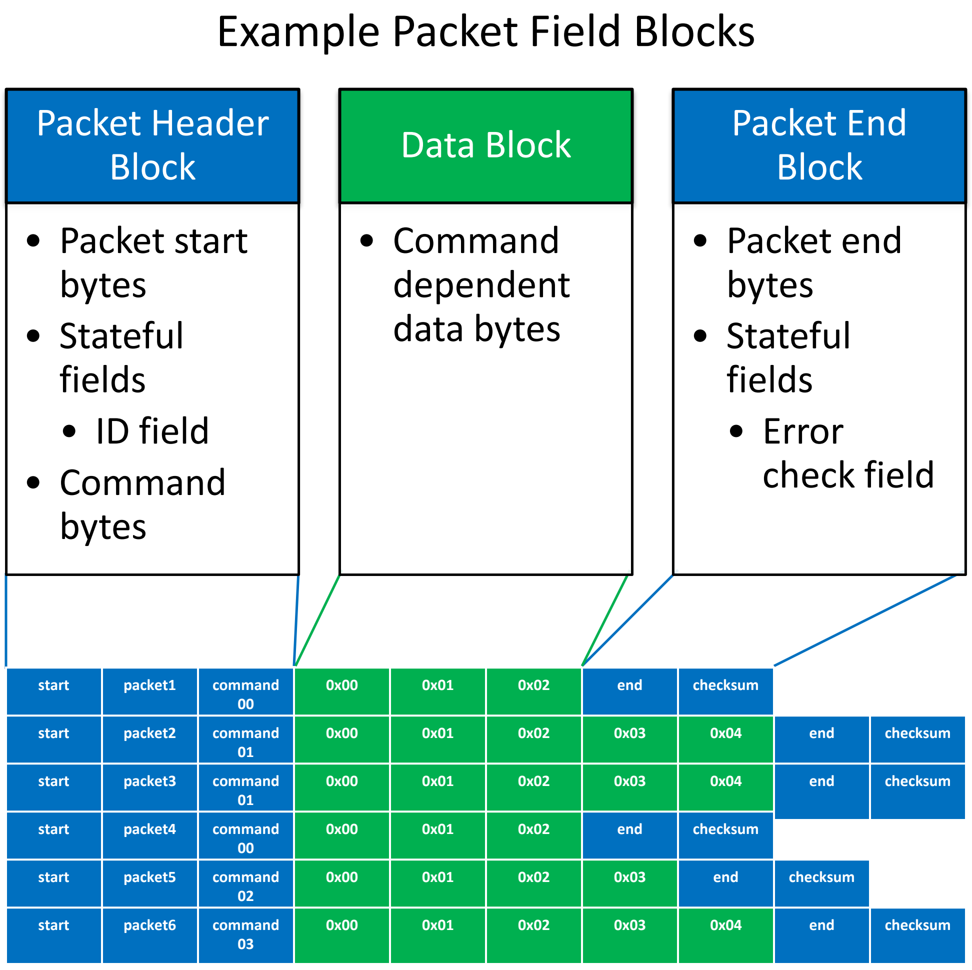


Figure 4.1: Example packet fields.

field occurrence rate. The field that optimizes the function is chosen as the start field. The capture is then divided into packets beginning with the identified start field.

Once grouped, the checksum field is accounted for by searching packets right to left for mismatches. Mismatches are identified as fields that change between the two baseline captures. If a mismatched field is found, the checksum identification algorithm computes a checksum of the packet data and compares it against the mismatched field value. Note that the algorithm computes checksums of various substrings within a packet as start bytes may not be included within the checksum computation. If the computed value and field value match for all packets in the capture, the field is identified as a checksum field.

Finally, other field mismatches are checked, such as the transaction number. The algorithm for remaining mismatches identifies fields by searching for the field value in the corresponding interface computer request packet. If a field of equivalent value is found at the same location for all mismatches within a capture, then the mismatch is grouped with the corresponding equivalent value. The protocol parser class diagrams are included in Appendix Figure C.3.

The contents of a firmware load including protocol and firmware bytes are visually represented in Figure 4.1. The figure shows example packet fields and their locations in an example capture. The stateful data fields applicable to DF1 are a two byte transaction number for packet identification and a block check character (BCC) checksum field. An escape byte of 0x10 is also used in the event end bytes occur within the data field [56]. The interface computer sends polling packets during power up while the PLC does not send packets until the power cycle is complete. Figure 4.2 identifies protocol and firmware bytes within a firmware capture. Each pixel represents a single byte in the capture. Approximate regions marked are connection initialization and connection close communications. Firmware upload data and protocol upload data is indicated as well, illustrating the approximate location and amount of overhead protocol data necessary for a firmware upload. The relative density of protocol bytes and the size of the file depict the potential for modification. The profile creation class diagrams are included in Appendix Figures C.2 and C.3. Pseudocode for tool protocol identification algorithms are included in Appendix A.

4.1.2   Data Emulation.   After the baseline is created, the tool can independently download and verify firmware data and communication data sent from an interface

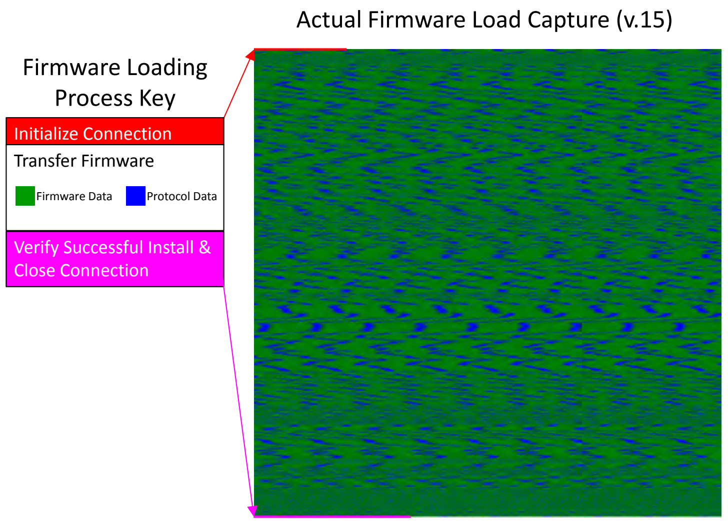


Figure 4.2: Protocol and firmware bytes in a version 15 capture.

computer. The tool uses the baseline profile to emulate the PLC, replaying the PLC's data with protocol modifications to communicate with and download firmware from the interface computer. The tool iteratively sends complete packets.

The emulator seeks to imitate the ControlFLASH loading process. Figure 4.3 shows a PLC capture excerpt after parsing, delineated by packet. Each pixel represents a single byte. The raw data as depicted in Figure 4.2 is analyzed and parsed into the displayed packets. The various field delineations are represented. During the connection initialization phase, the PLC communicates its model number and firmware revision number. The interface computer does not register a list of PLCs so the emulator can reply with any valid field values. Once loading begins, successive packets sent are acknowledgement packets to indicate successful receipt of data from the interface computer. When all data is received the PLC sends indication upon power cycle completion and closes the firmware upload connection.

The emulator allows the tool to be used independently of a PLC. Without the emulator feature, all captures require the tool to passively observe firmware loads between an interface computer and a PLC, which does not necessarily prevent malicious firmware from installing on the PLC. Independent data capture allows verification before installation, increasing the difficulty of a successful firmware attack. The PLC emulator class diagrams are included in Appendix Figure C.4

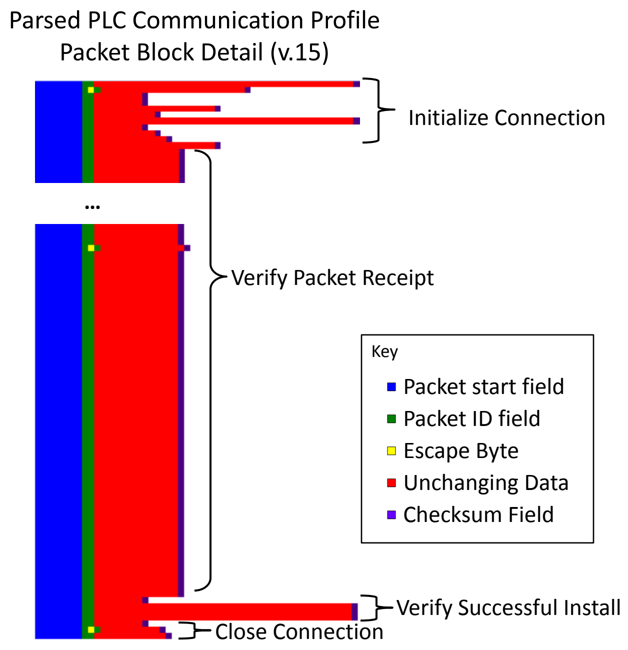


Figure 4.3: Subset of blocks created by the parsing and analysis process of a version 15 PLC data capture.

4.1.3 Firmware Verification. The two baseline firmware captures may have differing fields, if the protocol includes any stateful fields. The DF1 protocol, as outlined, does include stateful fields. Figure 4.4 illustrates example firmware capture baseline and verification test cases. Fields that are different from one baseline capture to the other are noted, and these fields are acceptable deviations for the capture in question as well. Escape bytes are also accounted for, if used. Beyond these expected deviations, the presence of mismatches between fields that are not baseline deviation fields, including lengthwise and bitwise mismatches, demonstrate the capture contains invalid data. The firmware verification classes are included in Appendix Figure C.5.

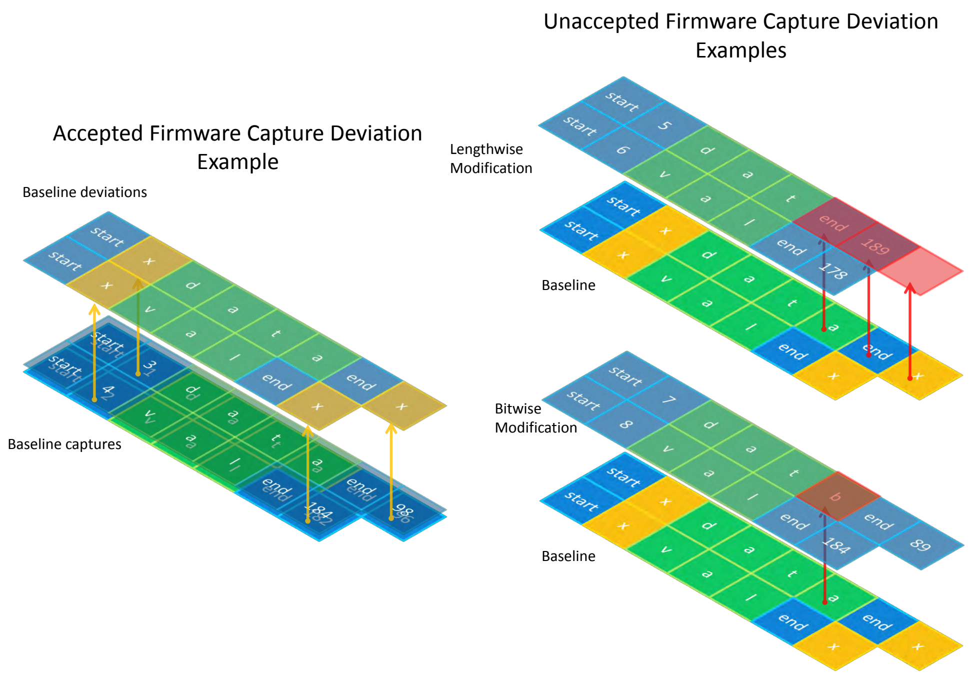


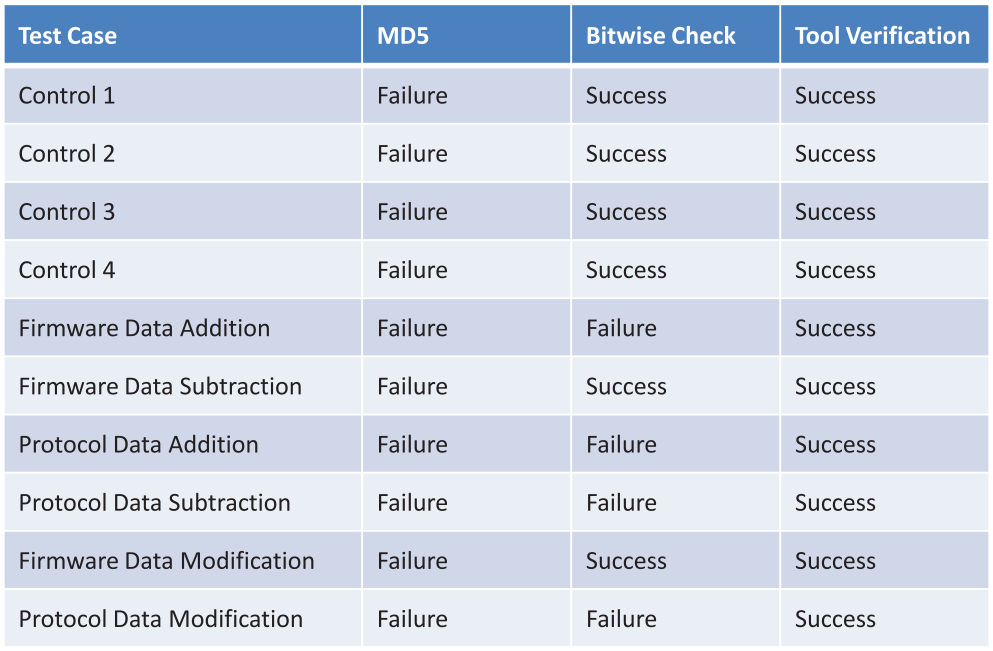
Figure 4.4: Example firmware capture verification cases.

4.2 Results

The results demonstrate the tool's successful baseline, analysis, and emulator capabilities. The tool's firmware verification routine successfully identified modified and unmodified firmware captures. Images of tool test case modifications are included in Appendix D and images of the tool performing test cases are included in Appendix C.6.

4.2.1 Baseline Capture and Analysis Results and Emulation Results. Each baseline capture and analysis routine test case successfully captured two loads and created a baseline profile without discrepancies or errors. The emulator successfully captured firmware loads utilizing each baseline version without discrepancies or errors.

4.2.2 Firmware Modification. Success and failure for each test case verification routine are noted in the Figure 4.5. The tool's verification routine correctly identified modified and unmodified firmware. The MD5 routine failed on all cases because it did not correctly identify control cases as unmodified. Bitwise check correctly identified the presence of firmware data bytes within a capture, but failed to recognize additional firmware data bytes, or protocol modifications.



Test Case	MD5	Bitwise Check	Tool Verification
Control 1	Failure	Success	Success
Control 2	Failure	Success	Success
Control 3	Failure	Success	Success
Control 4	Failure	Success	Success
Firmware Data Addition	Failure	Failure	Success
Firmware Data Subtraction	Failure	Success	Success
Protocol Data Addition	Failure	Failure	Success
Protocol Data Subtraction	Failure	Failure	Success
Firmware Data Modification	Failure	Success	Success
Protocol Data Modification	Failure	Failure	Success

Figure 4.5: Firmware verification routine results.

4.3 Analysis

The tool successfully verified unmodified captures, and correctly detected each modified capture. Additionally, the baseline capture and analysis successfully produced baseline profiles, and the emulator successfully imitated a PLC during firmware verification.

4.3.1 MD5 Hash, Bitwise Check, and Tool Verification Routines. The MD5 routine failed on all cases. Due to changing stateful fields, two functionally equivalent captures produce different hash values. The bitwise check routine failed to detect lengthwise additions and protocol modifications. These failures were anticipated, and demonstrate the shortcomings of a hash function or simple bitwise check. Indeed, checking captured data to ensure it contains firmware bytes is inadequate. The bitwise check case failures demonstrate that a verification routine must have protocol knowledge or it cannot perform a sufficiently thorough verification. The tool's firmware verification function correctly identifies the addition because it checks the firmware data as well as the protocol data.

The emulator ensures that packets are sent and received in the same manner as the baseline capture. It is critical that the tool verifies that the same firmware bytes are sent in a manner identical to the baseline. Without this verification, it is possible for an attacker to send packets that contain the same bytes as the firmware, but embed malicious data in the packet's protocol bytes. The captured data will still contain all firmware bytes, passing a bitwise check to see if the transfer contains all firmware data, but modifications of this type will fail a check that includes protocol verification. The bitwise firmware check test cases demonstrate the importance of the tool's emulation and verification functions - allowing not only the firmware data to be verified, but the communication protocol data as well.

4.4 Discussion on Limitations

4.4.1 Platform Limitations. The tool was tested and developed on a Windows 7 platform and requires two serial ports, or USB to serial adapters. This hardware requirement may be a limiting factor. Not all PLCs have serial firmware upload capabilities. Some PLCs may utilize proprietary loading devices or other communication standards as well. Additionally, The tool executes on a typical personal computer, which has the same vulnerabilities as the interface computer. Though it has the benefit of providing external verification to the interface computer's data transfer, the computer the tool is on may still be at risk of compromise. This risk can be mitigated if the tool is executed on a platform that has memory or processing constraints, limiting the possibility for platform modification (e.g., embedded device).

The tool may be detectable if the attacker is aware of the tool and has compromised the interface computer. The tool's emulator does not emulate the same timing of packets sent from the PLC, and the tool only responds in a manner imitating original PLC responses. It may not respond correctly to packets it does not expect, and an attacker can use this to their advantage. Note that an attack would have to detect these differences in a subtle manner, sending an unmodified load to the tool while continuing to send modified firmware to the PLC. However, the tool can still capture any differences when loading firmware directly to the PLC by using the tool's passive serial capture capability. Any attack would require extensive knowledge of the tool.

4.4.2 Development Limitations. Protocol pattern detection relies on the computer analyzing the data and deriving potential fields, based on the data. Although sufficient for the tested firmware, it is possible that these patterns may not always provide a conclusive profile for other protocol implementations. The current functionality is not a complete representation of all possible patterns that exist in today's protocols. The development focused on the DF1 protocol; further development is necessary before the tool can be considered platform independent.

4.5 Analysis Summary

Using the evaluation technique, the tool's baseline analysis and emulation routines were tested with firmware uploads, successfully creating baseline profiles and emulating PLC traffic to capture firmware loading data. The firmware verification routine successfully identified modification or no modification on all test cases. The MD5 routine failed on all cases due to inability to identify unmodified data, and the bitwise check failed to detect firmware data additions and protocol modifications. The tool verification routine's success rate demonstrates its tailored ability to recognize modified firmware.

5 Conclusions and Future Work

5.1 Conclusions

Critical infrastructure cyber security research and development is currently focused on the network layer. Critical infrastructure protection standards and best practices seek to reduce the potential for network attacks originating from the public internet because of the increasingly online nature of SCADA systems [54]. Many proposed and developed tools add encryption or intrusion detection systems to SCADA networks through field device modification or by a bump-in-the-wire device addition. While these tools may be successful at auditing malicious or atypical network traffic, all fail to mitigate malware originating from potentially trusted or non-network layer ingress vectors. Escaping the electronic perimeter mentality, this research seeks to expand SCADA system security to provide integrity and security below the network layer. PLC firmware is a highly vulnerable target because of its capability to control a PLC, lack of access control security, and lack of auditability.

The tool presented is a firmware emulation and validation tool intended to audit firmware loads from an interface computer to a PLC at the last externally electronically modifiable point. The tool is a novel application, designed to be used as a malware prevention and detection device for PLC firmware. The tool increases SCADA system security by creating a closed system with respect to the PLC firmware by validating inputs from the interface computer; once initiated in a secure state the PLC remains in a secure state as long as firmware inputs are valid. The tool can be set up to work with multiple PLCs, and once the communication baseline has been created, the tool operates independently of a PLC. The tool's ability to check both firmware and communication data increases the extent of firmware integrity assurance beyond that of a MD5 hash or simple bitwise firmware data checking routine.

The tool was developed and tested on a Windows personal computer. The tool successfully executed all test cases which consisted of testing the tool's emulation and verification routines with the Allen-Bradley FlexLogix 5434 PLC and Rockwell Software's ControlFLASH firmware upload program. The tool successfully demonstrates a proof of concept for a new paradigm of SCADA system cyber security tools focused on field device security as opposed to electronic perimeter security.

5.2 Impact

This tool is intended as a starting point for the development of a SCADA cyber security toolkit that focuses on vulnerabilities beyond the network layer. The tool is meant primarily for malicious cyber attack deterrence and detection. While the network layer does present possible cyber attack points, limiting defense to the network layer will not provide complete security. This tool seeks to expand the SCADA cyber defense toolkit by providing PLC centric security. Specifically, the adaptable baseline and PLC emulation functionality is operationally useful as it allows resource and manpower reduction while still providing firmware integrity assurance. Checking every PLC in the field is usually infeasible due to the large quantity and remote locations of PLCs. However, checking every firmware loading computer is a simpler task as these computers are likely more accessible. Instead of checking PLCs, checking every computer that loads PLC firmware is an operationally feasible problem and grants an increased degree of confidence that PLCs do not contain modified firmware.

Regardless of injection point, this tool captures firmware modifications implemented through the interface computer's firmware loading mechanism. The change from network centric security to tailored device security demonstrates a paradigm shift. Consider, for example, the scenario outlined in Figure 5.1. Vectors that utilize a non networked ingress point gain access to the SCADA system components including the interface computer. The interface computer has access to the PLC. Additional device security at the PLC layer audits data from sources inside the network security perimeter, detecting potential modifications. The tool presented exemplifies how this new paradigm can be applied to increase SCADA system and critical infrastructure security.

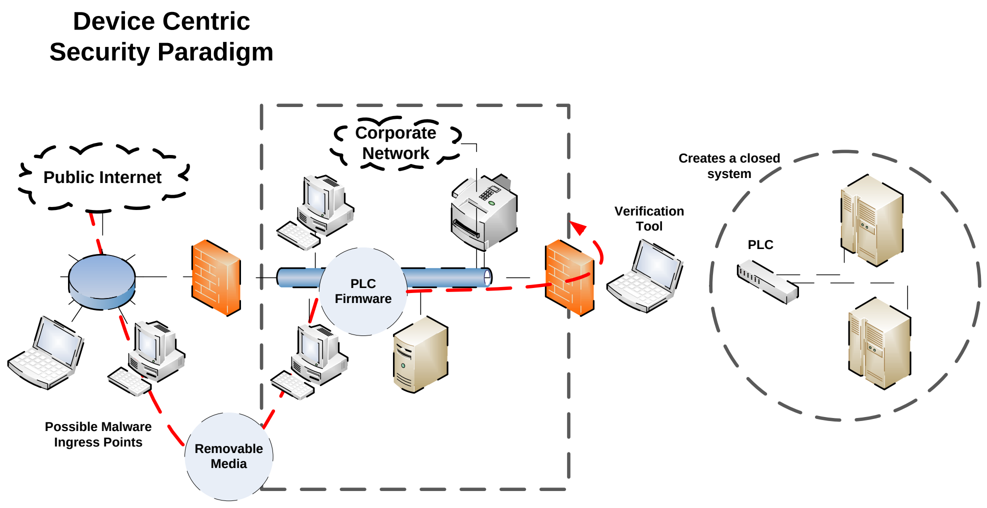


Figure 5.1: Device centric security.

5.2.1 Impact to Manufacturers. PLC manufacturers are considered trusted actors. Published SCADA security standards assume that software and hardware from a trusted company must be trusted as well. This can be dangerous when PLC technicians connect to a third party server to download firmware or software updates. Malicious modifications to the manufacturer's firmware version propagate to all end users who download firmware updates. PLC manufacturing companies can benefit from firmware verification because it grants them the ability to baseline and verify their own versions on a reoccurring basis. Additionally, the PLC manufacturers can produce a baseline profile for end users to compare against their own baseline. If the known good is available for inspection by the community, any incidents of detected firmware modification can be more quickly identified and reported, improving speed of mitigation and reducing potential malicious effects.

5.2.2 Impact for Industry. The majority of critical infrastructure is privatized, so the benefit of any proposed security measures must outweigh the cost if it is to be implemented. Expensive tools, developmental tools, or tools that require system reconfiguration can be difficult to justify on an already functioning system.

There have been no documented critical infrastructure cyber attacks targeting the United States. However, waiting until an attack to implement security measures retroactively could be dangerous considering the potential kinetic effects of a successful attack. A toolkit of simple but comprehensive tools focused on protecting SCADA system field devices that are relatively inexpensive and easily implementable would benefit the critical infrastructure industry.

5.2.3 Impact for Security Professionals. SCADA security best practices do not currently advocate any tools for field device security, and security teams such as ICS-CERT currently lack tools for SCADA specific malware detection. There is a need for SCADA specific malware prevention and detection tools. The firmware verification tool's primary use is to prevent malicious firmware from being uploaded onto a PLC. However, the tool can also be used as a response mechanism if a known good baseline is available. The tool can be used to check all possible interface computers and identify which, if any, computers were compromised. The tool's portability and versatility make it a prime candidate for operational implementation. Checking each computer versus checking each PLC reduces the problem complexity, decreasing system recovery time for security teams. Additionally, verifying each interface computer is sending unmodified firmware increases the level of security assurance with respect to PLC devices. This type of tailored assurance is not currently provided by network security tools. With further development, this tool could be instrumental in preventing malicious PLC firmware modifications on a variety of PLCs.

5.2.4 Potential Implementations. The tool has a variety of potential uses. The tool can be used to verify firmware integrity before applying firmware updates to PLCs. HMIs can also be checked to ensure they transfer unmodified firmware. The tool can audit communication between a PLC and interface computer during an update to identify potential malicious inclusions. If malware modifies firmware data, then the tool can pinpoint the infected computer. Additionally the tool's passive tap ability can capture serial communication data and log this data for future auditing.

5.2.5 Potential Weaknesses. An attacker attempting to circumvent the tool may discover its presence by checking packet response times. The tool runs faster than an actual PLC due to increased processor speed. Additionally, the tool only responds in a way as to mimic the PLC's expected response. An attacker could determine if the tool is being used by sending unexpected firmware packets and receiving an incorrect response. Note that any deviations beyond expected field differences will still be detected during a firmware load. Additionally, this tool does not account for firmware modifications that are manifested in ways other than through the firmware loading interface.

5.3 Future Work

5.3.1 Increased Testing. Testing was limited to one PLC model and one interface computer. Further testing on different models and manufacturers is needed to examine portability. Testing modified firmware also presents limiting factors. Allowing greater delays between loads may produce different baseline data. Currently a one upload delay is used; this delay can be lengthened in future testing.

5.3.2 Improved System Compatibility. Follow on work for firmware verification includes expanding functionality to work with multiple PLCs and SCADA system implementations. This includes research and development on various PLC protocols to create new protocol analysis patterns as well as adding functionality for system setups that use non-serial loading mechanisms. Ethernet is the next logical capture interface. Additionally, checksum routines are currently limited to the BCC function; cyclic redundancy checking or other routines would provide pertinent added functionality.

5.3.3 Increased Tool Security. The verification tool's platform can be modified to add security to the tool platform itself. The tool executes on a personal computer, so it is vulnerable to attack as well. To increase the tool's security, the tool can be ported to an embedded device, or used in conjunction with a field programmable gate array. These changes increase the tool's security by limiting its memory and computation capabilities.

5.3.4 Potential to Test All Possible PLCs. The emulator can be further developed to have greater functionality. The emulator currently only replays PLC data, so from the perspective of the interface computer, it appears that the same PLC is connected each time. The PLC identification response can be updated so that various PLC identification values are sent. If an attack is targeted at a specific PLC, then modifying these identification values may help uncover malicious system modifications. Iterating through possible values may trigger specific logic conditions. The tool can masquerade as the full range of PLCs without requiring increased equipment or resources by changing identification values.

5.3.5 Traffic Generation. The tool also has potential for PLC traffic generation as a penetration testing/fuzzing tool, or as a node in a honeynet setup. The tool applies patterns to captured data to generate valid packets used during a firmware load. The tool can be modified to generate PLC traffic for these alternative applications as well. The tool can capture and baseline data other than a firmware load, giving it different protocol knowledge and different replay ability. The tool can appear to be a variety of different systems, depending on the emulator baseline, without requiring additional resources for each new setup. Functionality can also be added to generate packets with minor deviations adhering to protocol rules by using evolutionary algorithms. Generated traffic can identify malicious attack or reconnaissance attempts. The tool can also generate packets designed for testing PLCs or other field devices. When used with a test PLC, the tool can iterate through valid packet combinations to uncover potentially insecure PLC actions or failure conditions.

5.3.6 Alternative PLC Layer Security. The tool can also be used to check communication data in settings other than firmware loads, such as during a logic program load or a program block value check. For example, Stuxnet used a modified DLL to hide a value stored on the PLC from the HMI software and end user [22]. If the function block value check communication data was baselined after initial install, the Stuxnet modifications would be detected.

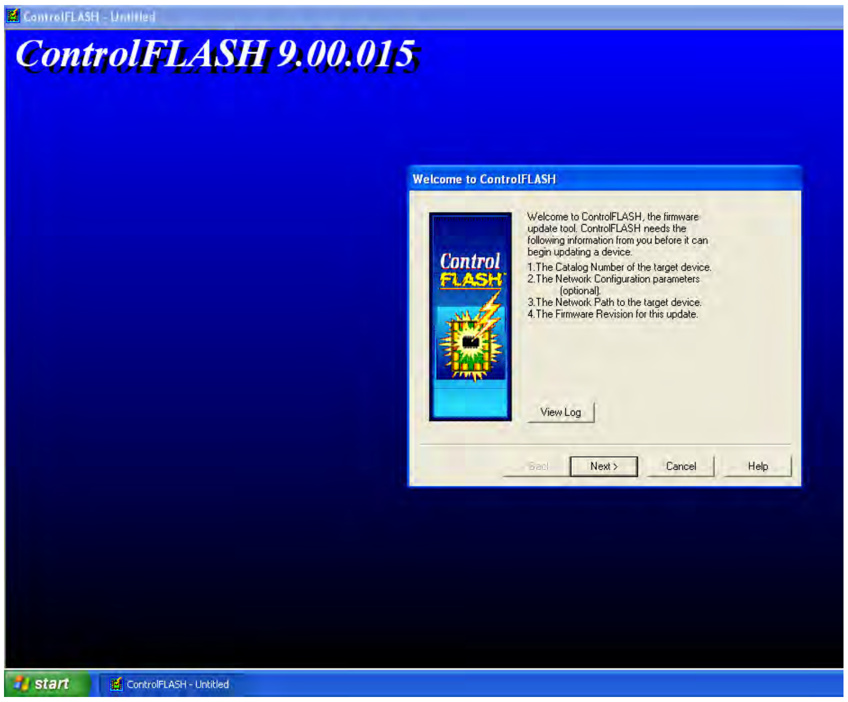
5.4 Concluding Remarks

Creating security tools specific to SCADA systems is necessary to maintain and build trust in critical infrastructure systems. The primary goal of validating firmware extends beyond ensuring known good firmware is loaded onto a PLC – it also helps create a closed system with respect to the PLC. The PLC has the highest level of local control over a SCADA system, so it is critical that controllers are verified on basic hardware and software levels before security measures can be effectively applied to higher levels. Firmware is the lowest electronically modifiable level of many PLCs. Indeed, firmware validation is the first logical step when considering electronic security.

The firmware verification tool captures serial data during firmware uploads and verifies captured upload data against a known good baseline. The tool has PLC emulation functionality and can analyze firmware without the presence of a PLC. While serial data capture, data verification, and emulation are not new ideas individually, the tool combines these ideas in a novel manner tailored to SCADA system security. The verification tool offers a novel approach tailored for SCADA security because it requires no system modifications or additions and does not affect the production system. Additionally, implementing the verification tool does not introduce attack input vectors to the PLC because the tool it is not physically wired to exchange communication with the PLC. The verification tool is a viable option for increasing PLC firmware security.

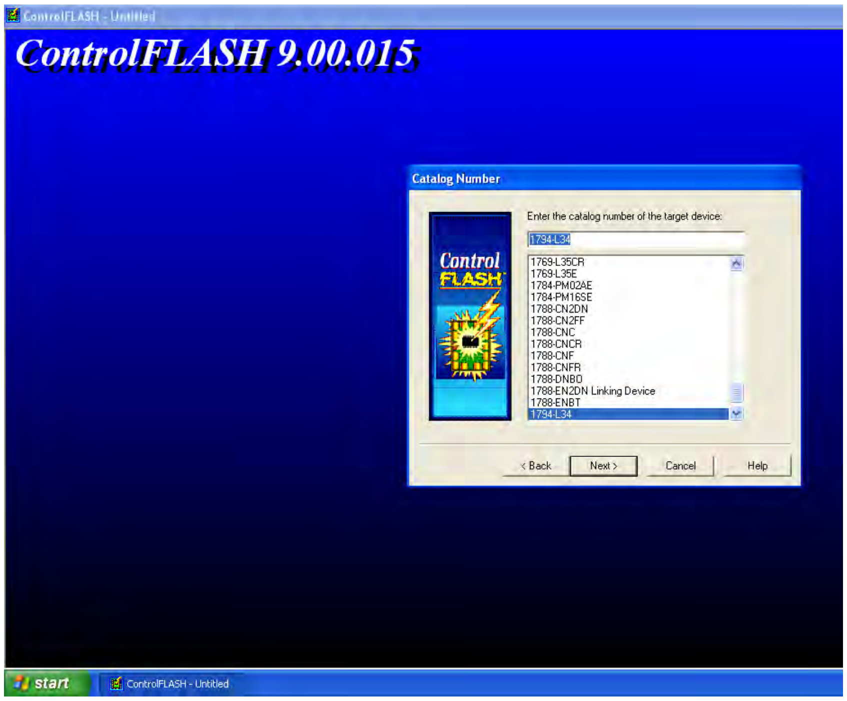
Appendix A: Typical ControlFLASH Firmware Load Process

The following screenshots outline a typical ControlFLASH firmware load process to the Allen-Bradley FlexLogix 5434 PLC.



Welcome to ControlFLASH, the firmware update tool. ControlFLASH needs the following information from you before it can begin updating a device.
1.The Catalog Number of the target device.
2.The Network Configuration parameters (optional).
3.The Network Path to the target device.
4.The Firmware Revision for this update.

Figure A.1: Open ControlFLASH.

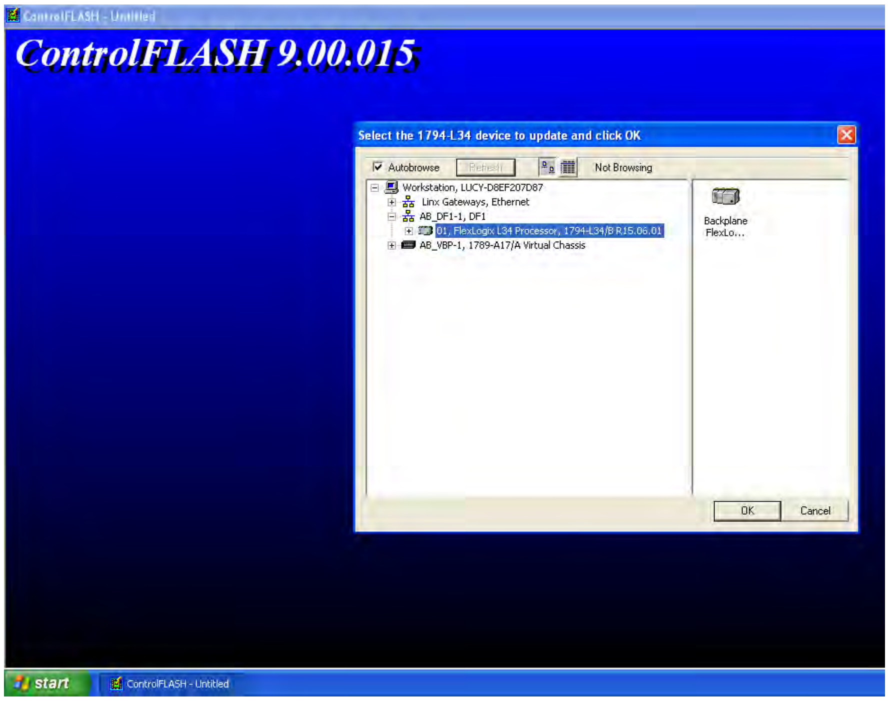


Enter the catalog number of the target device:

1794-L34

1769-L35CR
1769-L35E
1784-PM02AE
1784-PM16SE
1788-CN2DN
1788-CN2FF
1788-CNC
1788-CNCR
1788-CNF
1788-CNFR
1788-DNBO
1788-EN2DN Linking Device
1788-ENBT
1794-L34

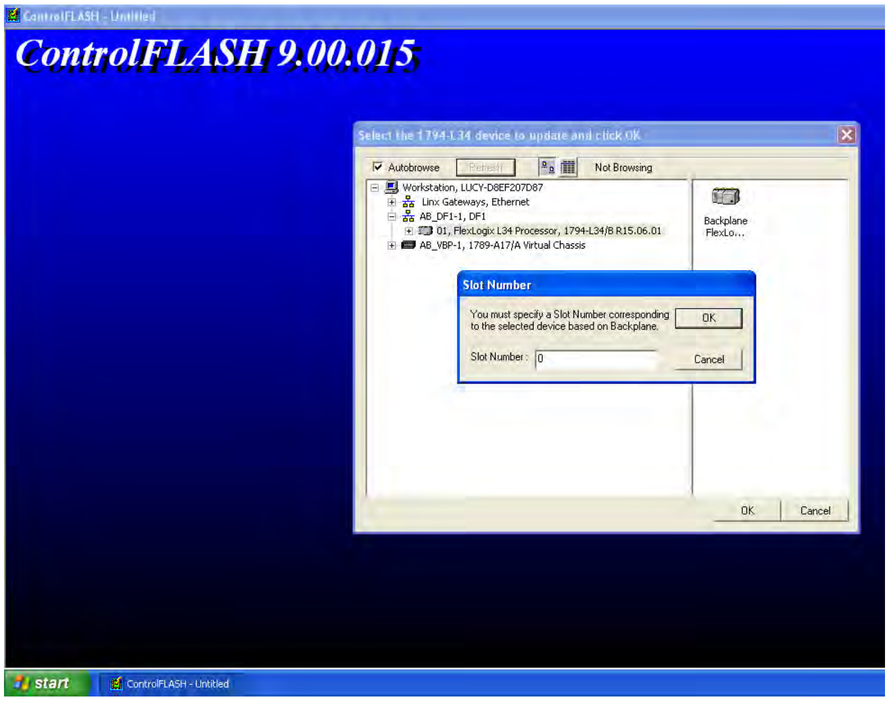
Figure A.2: Select PLC type.



Select the 1794-L34 device to update and click OK

Workstation, LUCY-D8EF207D87
Linx Gateways, Ethernet
AB_DF1-1, DF1
01, FlexLogix L34 Processor, 1794-L34/B R15.06.01
AB_VBP-1, 1789-A17/A Virtual Chassis

Figure A.3: Select PLC from device network.



Select the 1794-L34 device to update and click OK

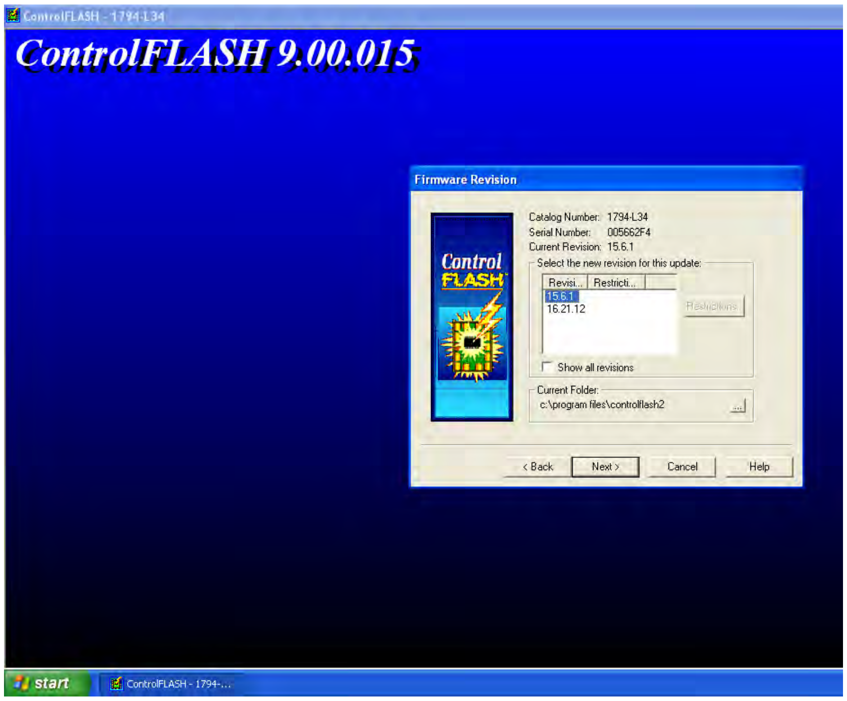
Workstation, LUCY-D8EF207D87
Linx Gateways, Ethernet
AB_DF1-1, DF1
01, FlexLogix L34 Processor, 1794-L34/B R15.06.01
AB_VBP-1, 1789-A17/A Virtual Chassis

Slot Number

You must specify a Slot Number corresponding to the selected device based on Backplane.

Slot Number : 0

Figure A.4: Enter PLC designator number.



Firmware Revision

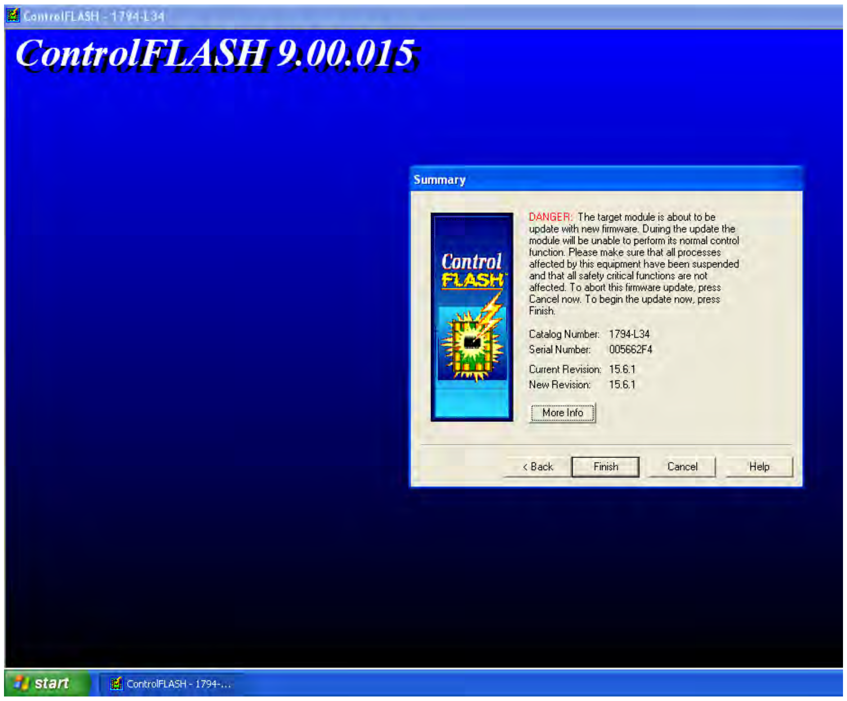
Catalog Number: 1794-L34
Serial Number: 005662F4
Current Revision: 15.6.1
Select the new revision for this update:

Revisi...	Restricti...
15.6.1	
16.21.12	

Show all revisions

Current Folder:
c:\program files\controlflash2

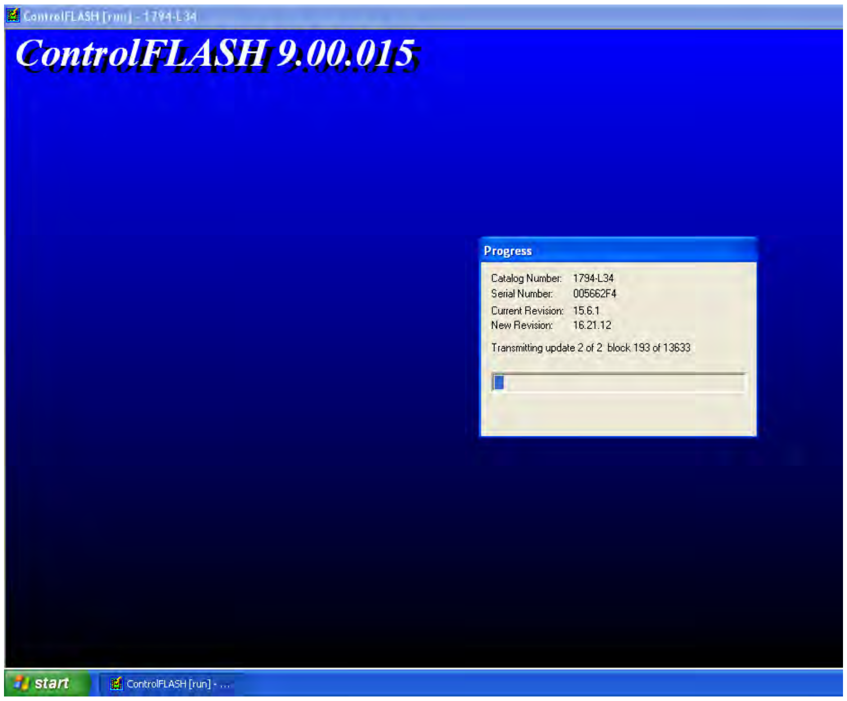
Figure A.5: Select firmware revision number.



DANGER: The target module is about to be update with new firmware. During the update the module will be unable to perform its normal control function. Please make sure that all processes affected by this equipment have been suspended and that all safety critical functions are not affected. To abort this firmware update, press Cancel now. To begin the update now, press Finish.

Catalog Number: 1794-L34
Serial Number: 005662F4
Current Revision: 15.6.1
New Revision: 15.6.1

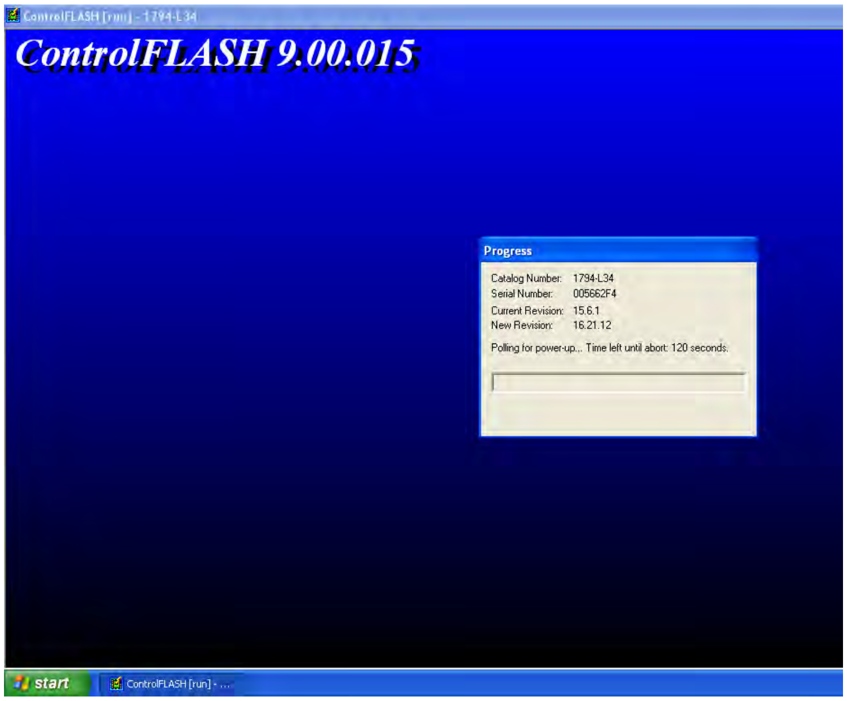
Figure A.6: Begin upload.



Catalog Number: 1794-L34
Serial Number: 005662F4
Current Revision: 15.6.1
New Revision: 16.21.12

Transmitting update 2 of 2 block 193 of 13633

Figure A.7: Uploading in progress.


ControlFLASH [run] - 1794-L34

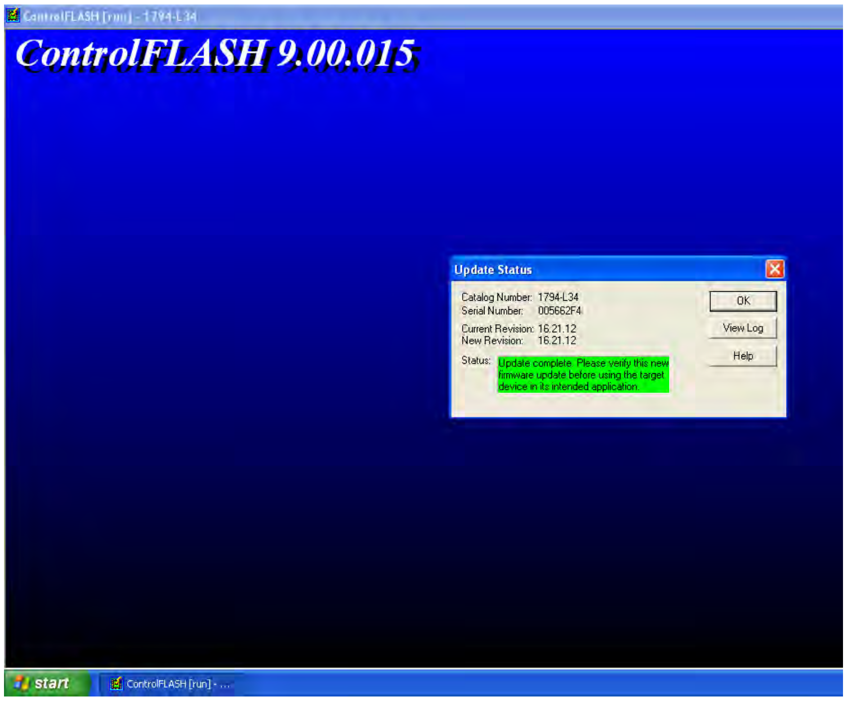
ControlFLASH 9.00.015

Progress

Catalog Number: 1794-L34
Serial Number: 005662F4
Current Revision: 15.6.1
New Revision: 16.21.12

Polling for power-up... Time left until abort: 120 seconds.

Figure A.8: Power cycle upon upload.


ControlFLASH [run] - 1794-L34

ControlFLASH 9.00.015

Update Status

Catalog Number: 1794-L34
Serial Number: 005662F4
Current Revision: 16.21.12
New Revision: 16.21.12
Status: Update complete. Please verify this new firmware update before using the target device in its intended application.

Figure A.9: Successful Install.

Appendix B: Pseudocode

The following algorithms abstract the tool's routines and functionality.

B.1 Data Capture

```
Input: $plcSerialLine$, $computerSerialLine$
Output: $plcData$, $computerData$
Thread $plcThread$ = new Thread($plcSerialLine$, $plcData$)
Thread $computerThread$ = new Thread($computerSerialLine$, $computerData$)
Thread Function():
begin
    while $receiving$ do
        if $mySerialLine.hasData$ then
            $myData$.add($mySerialLine$.data)
        end
    end
end
```

Figure B.1: Data Capture Algorithm

B.2 Protocol Analysis

```
Input: captureSet $plcData[2]$, captureSet $computerData[2]$
Output: $protocolProfile$
foreach captureSet dataSet do group captures into packets for both the PLC and computer
    GroupIntoBlocks($dataSet.capture1$, $dataSet.capture2$)
end
SavePackets()
SaveMismatches()
```

Figure B.2: Protocol Analysis Algorithm

```
Input: captureSet $plcData[2]$, captureSet $computerData[2]$
Output:
GroupIntoBlocks($capture1$, $capture2$):
begin
    List $possibleValues$ = FindMostUsedByteValues($capture1$)
    ByteList $startBytes = possibleValues[0]$
    int $usageDensity$ = FindNumOccurencesInCapture($capture1$, $startBytes$)
    $optimal$ = false
    while !optimal do create a start byte string of maximal length
        List $possibleValues$ = FindMostUsedByteValues($capture1$, $startBytes$)
        $startBytes = startBytes + possibleValues[0]$
        $newDensity = startBytes$.length +
        FindNumOccurencesInCapture($capture1$, $startBytes$).normalize
        if newDensity greater than usageDensity then
            $usageDensity = newDensity$
        else
            $optimal$ = true
            $startBytes$.removeLast()
        end
    end
    $dataSet$.setPacketBlocks($startBytes$)
end
```

Figure B.3: Protocol Analysis Block Grouping Algorithm

```
Input: captureSet plcData[2], captureSet computerData[2]
Output:
escapeByte = CheckForEscapeByte(plcData)
ResolvePLCMismatches():
begin
    foreach packet1, packet2 in plcData[1], plcData[2] do resolve mismatches
        if packet1 != packet2 and !checksum then
            checksum = CheckMismatchAsChecksum(packet1, packet2,
            escapeByte)
            if checksum then resolve for all packets
                SetChecksumForAllPackets(plcData[1], plcData[2])
            end
        elsecheck for data in equiv computer packet
            CheckIfRepeatedFromComputerBlocks(computerData[1], packet1)
        end
    end
end
```

Figure B.4: Protocol Analysis Mismatch Resolving Algorithm

```
Input: plcData
Output: bool escapeByte
CheckForEscapeByte(plcData):
begin
    List possibleValues = FindMostUsedByteValues(plcData)
    foreach possibleValues in plcData do see if escape strings work
        if plcData[1].applyEscapeByte(escapeByte) is more similar to
        plcData[2].applyEscapeByte(escapeByte) then
            return true
        end
    end
    return false
end
```

Figure B.5: Protocol Analysis Escape Byte Check Algorithm

B.3 PLC Emulator

```
Input: $computerSerialLine$, $protocolProfile$
Output: $computerData$
while $receiving$ do
    $computerData$.addData($computerSerialLine$.data)
    if $computerData.receivedFullPacket$ then a packet has been received
        $computerSerialLine$.write(produceNextPacket($protocolProfile$,
        $computerData$))
    end
end
return $computerData$
```

Figure B.6: PLC Emulator Algorithm

```
Input: $protocolProfile$, $computerData$
Output: $packet$
ProduceNextPacket($protocolProfile$, $computerData$):
begin
    $protocolProfile$.resolveMismatchesSoFar($computerData$)
    $packet$ = $protocolProfile$.nextPacket()
    $protocolProfile$.applyChecksum($packet$)
    $protocolProfile$.applyEscapeBytes($packet$)
    return $packet$
end
```

Figure B.7: PLC Emulator Packet Production Algorithm

B.4 Firmware Verifier

```
Input: baselineData[2], receivedData
Output: valid
List baselineMismatches = getMismatches(baselineData[1], baselineData[2])
List newMismatches = getMismatches(baselineData[1], receivedData)
if newMismatches == null then function returned error
    valid = false
    return valid
end
foreach mismatch in newMismatches do ensure all mismatches are accounted for
    if !baselineMismatches.contains(mismatch) then mismatch not expected
        valid = false
    end
end
return valid
```

Figure B.8: Firmware Verification Algorithm

```
Input: data1, data2
Output: mismatches[]
getMismatches(data1, data2):
begin
    prevByte1 = data1.readByte()
    prevByte2 = data2.readByte()
    location = 1
    while !data1.atEnd do
        if data2.atEnd then not enough bytes in data2
            return null
        end
        byte1 = data1.readByte()
        byte2 = data2.readByte()
        if byte1 != byte2 then data mismatch
            if prevByte1 == escapeByte && byte1 == escapeByte && prevByte1
            != prevByte2 then
                data2.position–
                location–
                prevByte1 = byte1
            end
            else if previousByte2 == escapeByte && byte2 == escapeByte &&
            prevByte1 != prevByte2 then
                data1.position–
                location–
                prevByte2 = byte2
            end
        else
            mismatches.add(location)
        end
        location++
        prevByte1 = byte1
        prevByte2 = byte2
    end
    if !data2.atEnd then too many bytes in data2
        return null
    end
end
```

Figure B.9: Firmware Verification Mismatch Algorithm

Appendix C: Tool Class Diagrams

The following class diagrams display the tool's code classes and their members and functions.

C.1 Main Class and Helper Classes

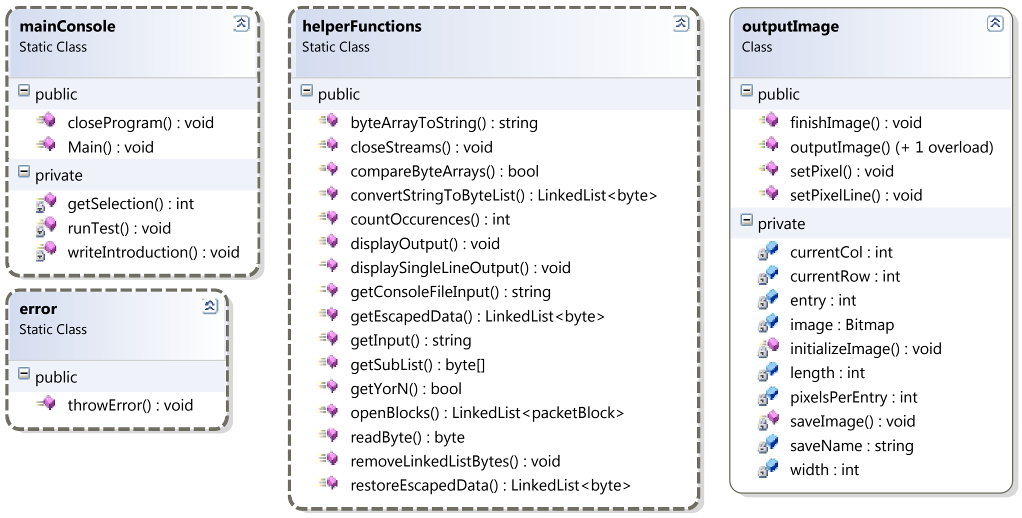


Figure C.1: Main Class and Helper Classes.

C.2 Passive Serial Capture Classes

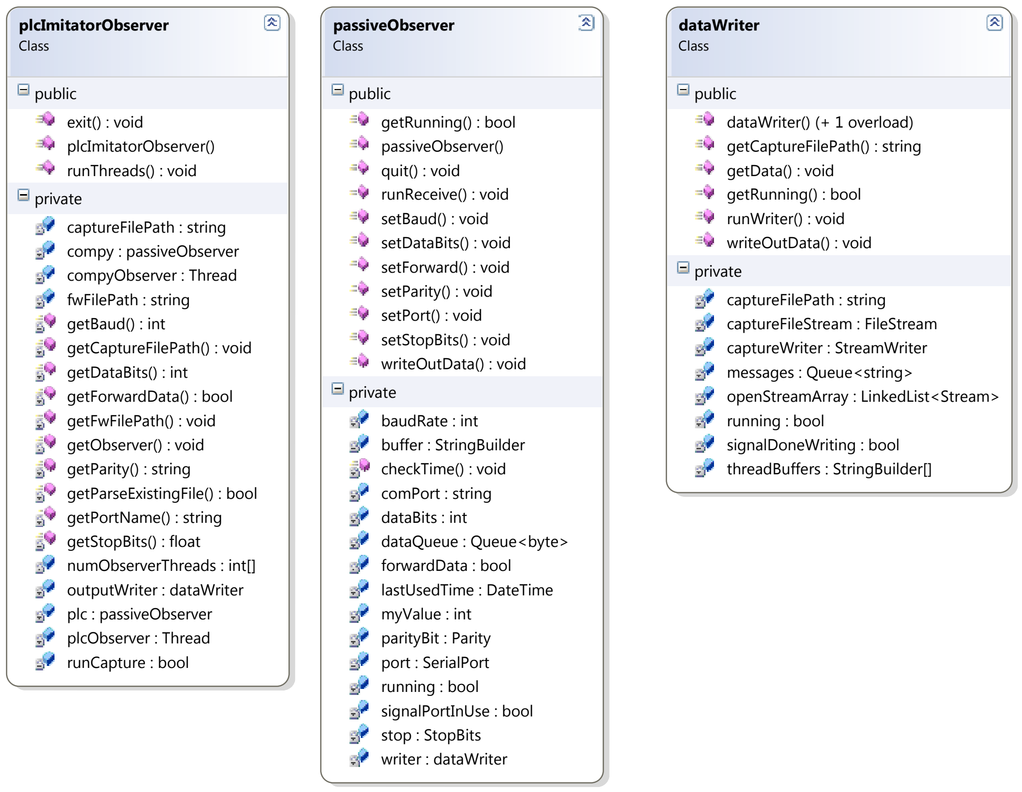
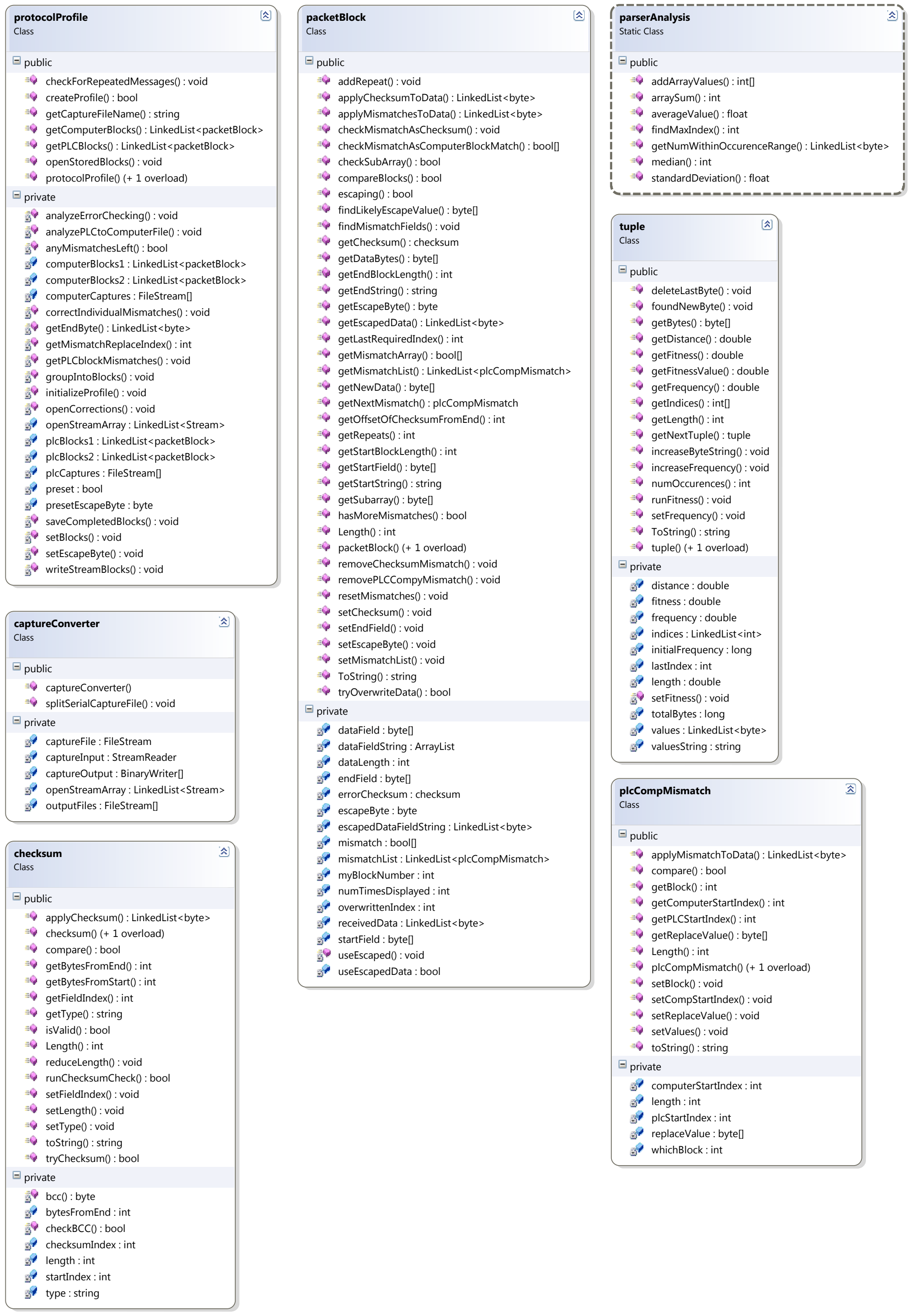


Figure C.2: Passive Serial Capture Classes.

C.3 Serial Capture Analysis and Profile Creation Classes



```
protocolProfile
Class
public
  checkForRepeatedMessages() : void
  createProfile() : bool
  getCaptureFileName() : string
  getComputerBlocks() : LinkedList<packetBlock>
  getPLCBlocks() : LinkedList<packetBlock>
  openStoredBlocks() : void
  protocolProfile() (+ 1 overload)
private
  analyzeErrorChecking() : void
  analyzePLCtoComputerFile() : void
  anyMismatchesLeft() : bool
  computerBlocks1 : LinkedList<packetBlock>
  computerBlocks2 : LinkedList<packetBlock>
  computerCaptures : FileStream[]
  correctIndividualMismatches() : void
  getEndByte() : LinkedList<byte>
  getMismatchReplaceIndex() : int
  getPLCblockMismatches() : void
  groupIntoBlocks() : void
  initializeProfile() : void
  openCorrections() : void
  openStreamArray : LinkedList<Stream>
  plcBlocks1 : LinkedList<packetBlock>
  plcBlocks2 : LinkedList<packetBlock>
  plcCaptures : FileStream[]
  preset : bool
  presetEscapeByte : byte
  saveCompletedBlocks() : void
  setBlocks() : void
  setEscapeByte() : void
  writeStreamBlocks() : void

captureConverter
Class
public
  captureConverter()
  splitSerialCaptureFile() : void
private
  captureFile : FileStream
  captureInput : StreamReader
  captureOutput : BinaryWriter[]
  openStreamArray : LinkedList<Stream>
  outputFiles : FileStream[]

checksum
Class
public
  applyChecksum() : LinkedList<byte>
  checksum() (+ 1 overload)
  compare() : bool
  getBytesFromEnd() : int
  getBytesFromStart() : int
  getFieldIndex() : int
  getType() : string
  isValid() : bool
  Length() : int
  reduceLength() : void
  runChecksumCheck() : bool
  setFieldIndex() : void
  setLength() : void
  setType() : void
  toString() : string
  tryChecksum() : bool
private
  bcc() : byte
  bytesFromEnd : int
  checkBCC() : bool
  checksumIndex : int
  length : int
  startIndex : int
  type : string

packetBlock
Class
public
  addRepeat() : void
  applyChecksumToData() : LinkedList<byte>
  applyMismatchesToData() : LinkedList<byte>
  checkMismatchAsChecksum() : void
  checkMismatchAsComputerBlockMatch() : bool[]
  checkSubArray() : bool
  compareBlocks() : bool
  escaping() : bool
  findLikelyEscapeValue() : byte[]
  findMismatchFields() : void
  getChecksum() : checksum
  getDataBytes() : byte[]
  getEndBlockLength() : int
  getEndString() : string
  getEscapeByte() : byte
  getEscapedData() : LinkedList<byte>
  getLastRequiredIndex() : int
  getMismatchArray() : bool[]
  getMismatchList() : LinkedList<plcCompMismatch>
  getNewData() : byte[]
  getNextMismatch() : plcCompMismatch
  getOffsetOfChecksumFromEnd() : int
  getRepeats() : int
  getStartBlockLength() : int
  getStartField() : byte[]
  getStartString() : string
  getSubarray() : byte[]
  hasMoreMismatches() : bool
  Length() : int
  packetBlock() (+ 1 overload)
  removeChecksumMismatch() : void
  removePLCCompyMismatch() : void
  resetMismatches() : void
  setChecksum() : void
  setEndField() : void
  setEscapeByte() : void
  setMismatchList() : void
  ToString() : string
  tryOverwriteData() : bool
private
  dataField : byte[]
  dataFieldString : ArrayList
  dataLength : int
  endField : byte[]
  errorChecksum : checksum
  escapeByte : byte
  escapedDataFieldString : LinkedList<byte>
  mismatch : bool[]
  mismatchList : LinkedList<plcCompMismatch>
  myBlockNumber : int
  numTimesDisplayed : int
  overwrittenIndex : int
  receivedData : LinkedList<byte>
  startField : byte[]
  useEscaped() : void
  useEscapedData : bool

parserAnalysis
Static Class
public
  addArrayValues() : int[]
  arraySum() : int
  averageValue() : float
  findMaxIndex() : int
  getNumWithinOccurenceRange() : LinkedList<byte>
  median() : int
  standardDeviation() : float

tuple
Class
public
  deleteLastByte() : void
  foundNewByte() : void
  getBytes() : byte[]
  getDistance() : double
  getFitness() : double
  getFitnessValue() : double
  getFrequency() : double
  getIndices() : int[]
  getLength() : int
  getNextTuple() : tuple
  increaseByteString() : void
  increaseFrequency() : void
  numOccurences() : int
  runFitness() : void
  setFrequency() : void
  ToString() : string
  tuple() (+ 1 overload)
private
  distance : double
  fitness : double
  frequency : double
  indices : LinkedList<int>
  initialFrequency : long
  lastIndex : int
  length : double
  setFitness() : void
  totalBytes : long
  values : LinkedList<byte>
  valuesString : string

plcCompMismatch
Class
public
  applyMismatchToData() : LinkedList<byte>
  compare() : bool
  getBlock() : int
  getComputerStartIndex() : int
  getPLCStartIndex() : int
  getReplaceValue() : byte[]
  Length() : int
  plcCompMismatch() (+ 1 overload)
  setBlock() : void
  setCompStartIndex() : void
  setReplaceValue() : void
  setValues() : void
  toString() : string
private
  computerStartIndex : int
  length : int
  plcStartIndex : int
  replaceValue : byte[]
  whichBlock : int
```

Figure C.3: Serial Capture Analysis and Profile Creation Classes.

C.4 PLC Emulator Classes

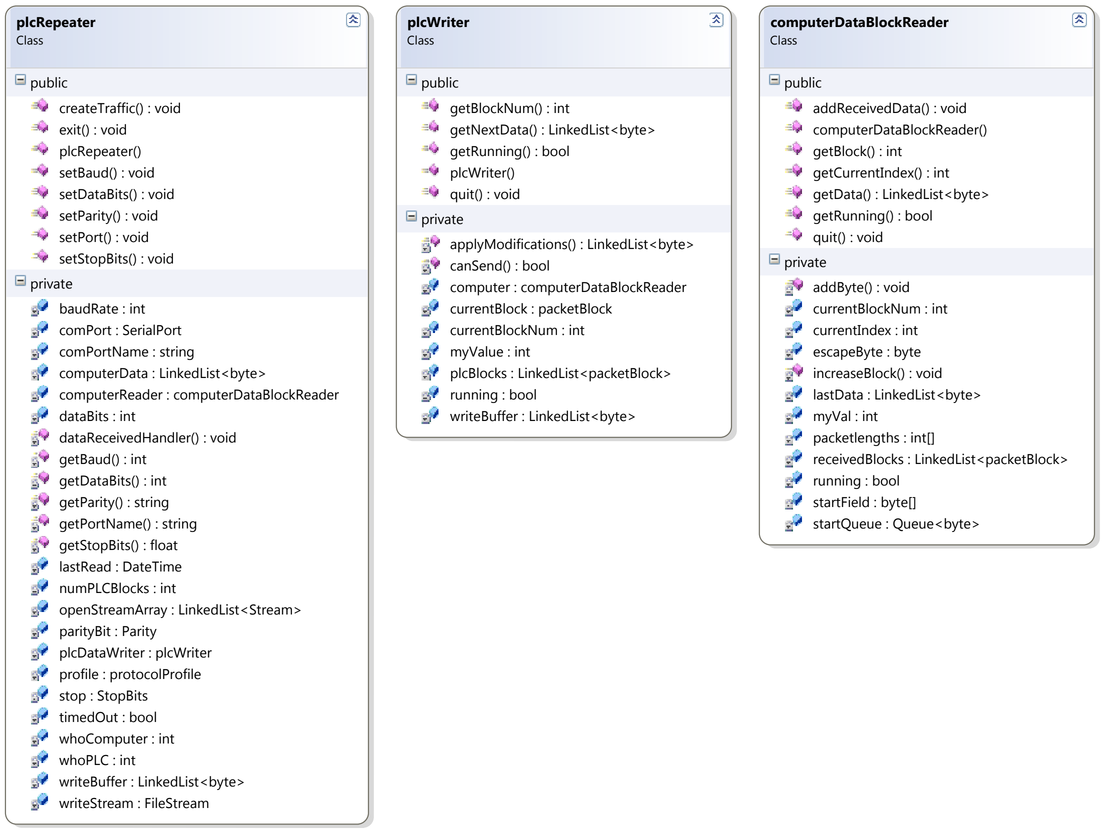


Figure C.4: PLC Emulator Classes.

C.5 Captured Firmware Verification Classes

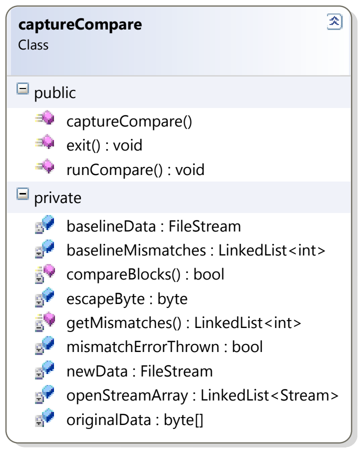


Figure C.5: Captured Firmware Verification Classes.

C.6 Firmware Parse and Check Classes

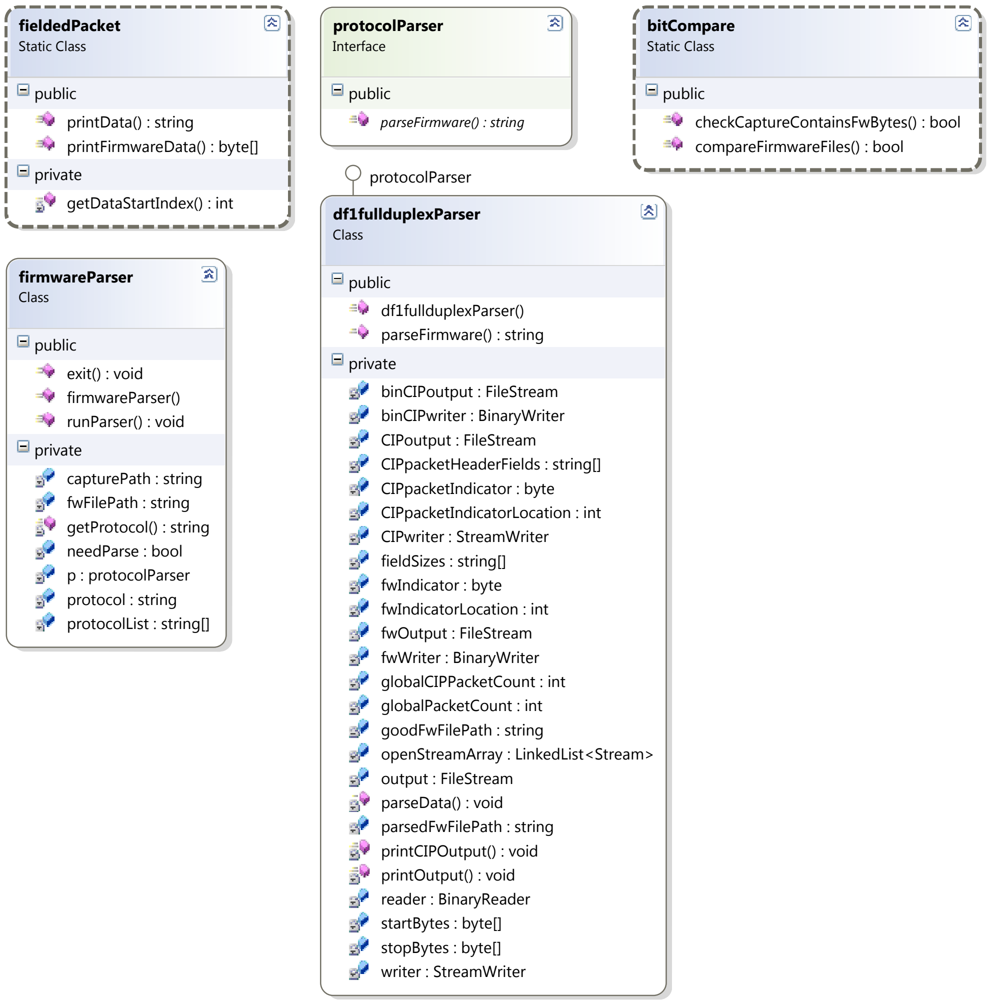
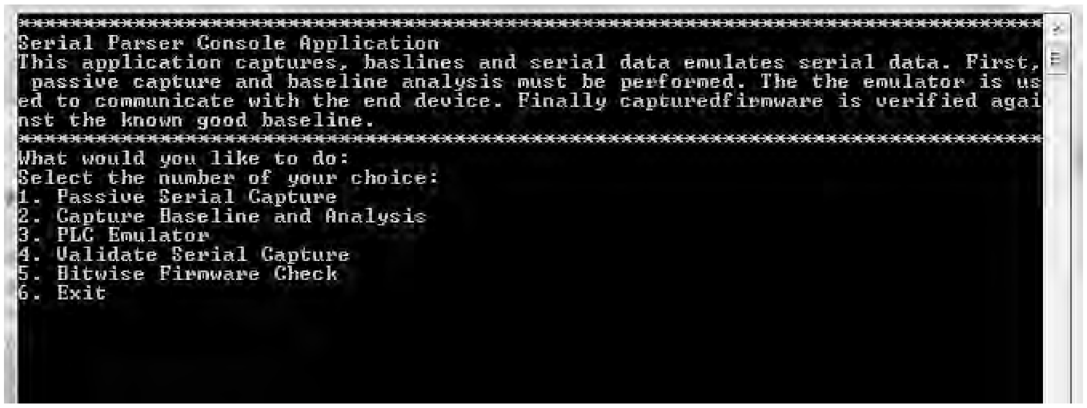


Figure C.6: Firmware Parse and Check Classes.

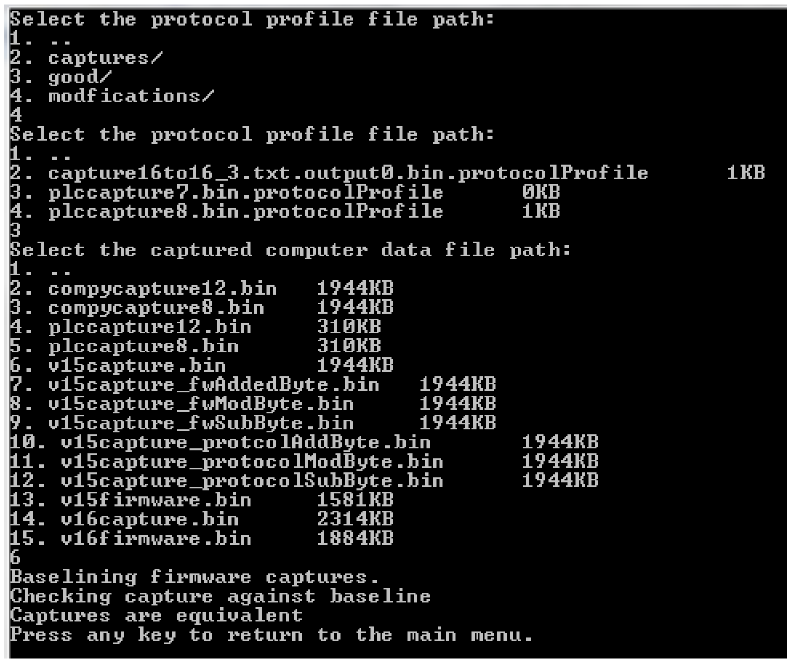
Appendix D: Tool Performing Test Cases

The following screenshots demonstrate use of the tool's firmware verification routine.



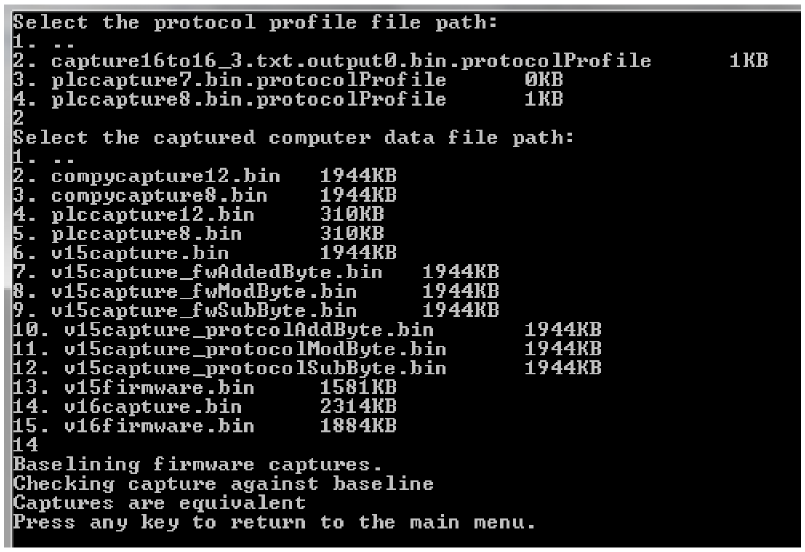
```
******************************************************************************
Serial Parser Console Application
This application captures, baslines and serial data emulates serial data. First,
 passive capture and baseline analysis must be performed. The the emulator is us
ed to communicate with the end device. Finally capturedfirmware is verified agai
nst the known good baseline.
******************************************************************************
What would you like to do:
Select the number of your choice:
1. Passive Serial Capture
2. Capture Baseline and Analysis
3. PLC Emulator
4. Validate Serial Capture
5. Bitwise Firmware Check
6. Exit
```

Figure D.1: Tool function selection screen.



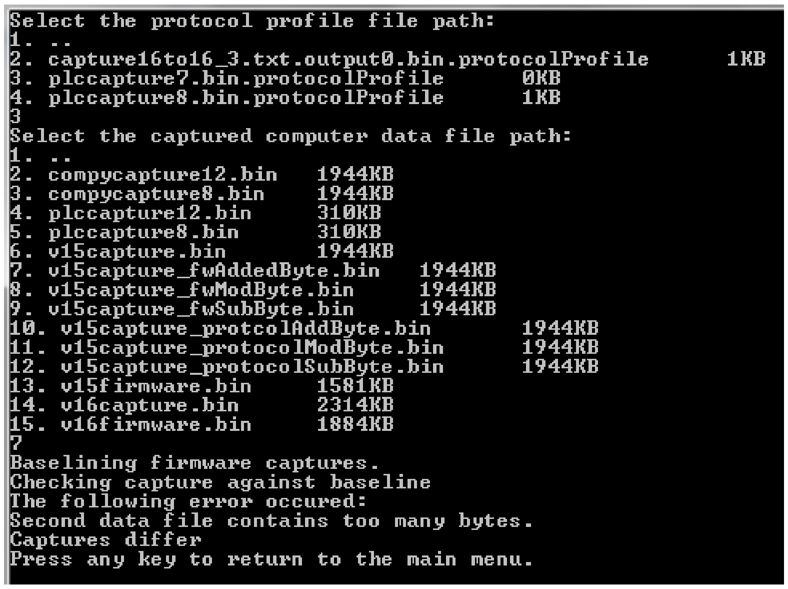
```
Select the protocol profile file path:
1. ..
2. captures/
3. good/
4. modfications/
4
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
6
Baselining firmware captures.
Checking capture against baseline
Captures are equivalent
Press any key to return to the main menu.
```

Figure D.2: Version 15 capture verification test.



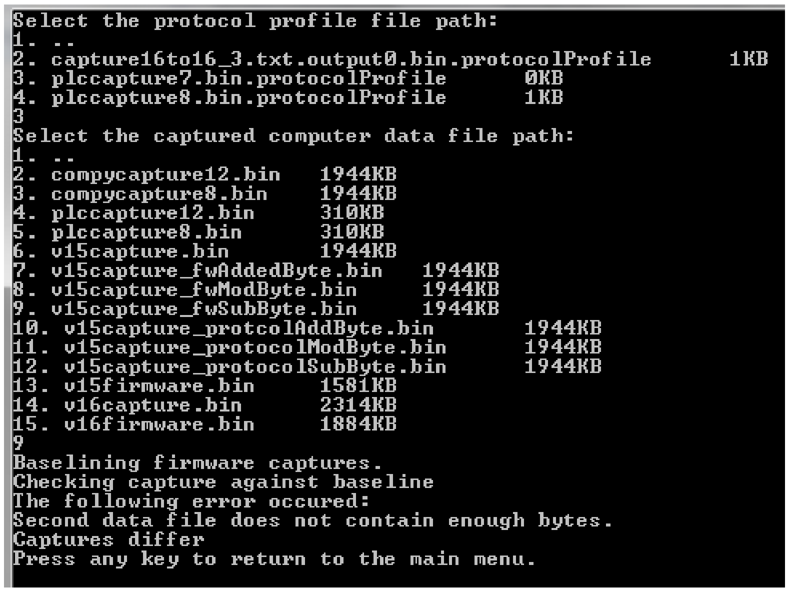
```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
2
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
14
Baselining firmware captures.
Checking capture against baseline
Captures are equivalent
Press any key to return to the main menu.
```

Figure D.3: Version 16 capture verification test.



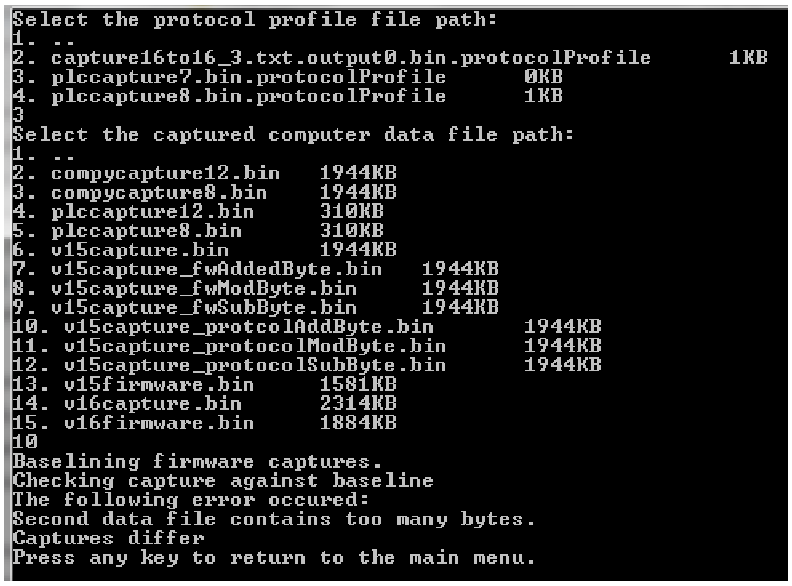
```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
7
Baselining firmware captures.
Checking capture against baseline
The following error occured:
Second data file contains too many bytes.
Captures differ
Press any key to return to the main menu.
```

Figure D.4: Byte added to firmware verification test.



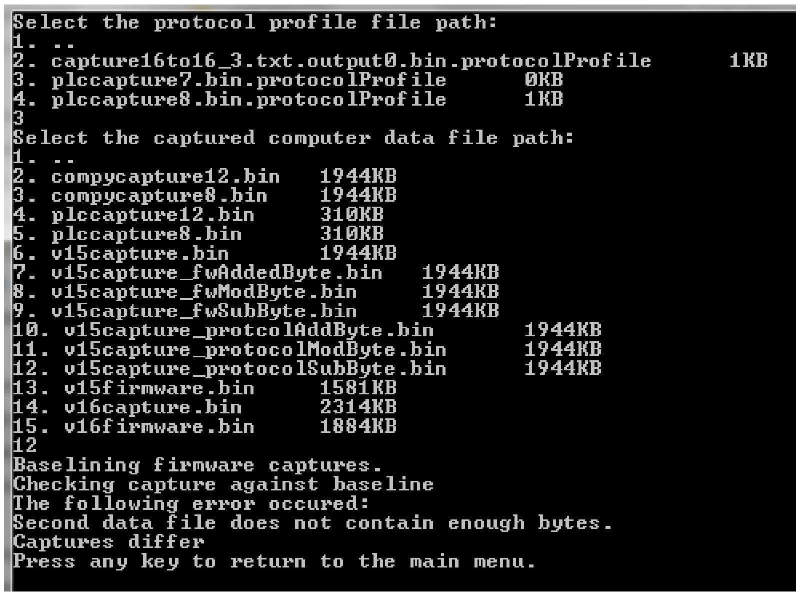
```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
9
Baselining firmware captures.
Checking capture against baseline
The following error occured:
Second data file does not contain enough bytes.
Captures differ
Press any key to return to the main menu.
```

Figure D.5: Byte removed from firmware verification test.



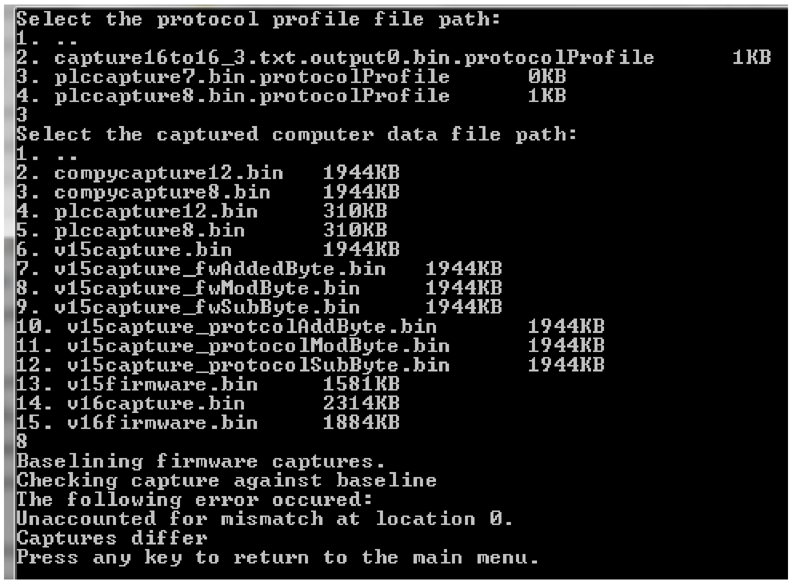
```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
10
Baselining firmware captures.
Checking capture against baseline
The following error occured:
Second data file contains too many bytes.
Captures differ
Press any key to return to the main menu.
```

Figure D.6: Byte added to protocol data verification test.



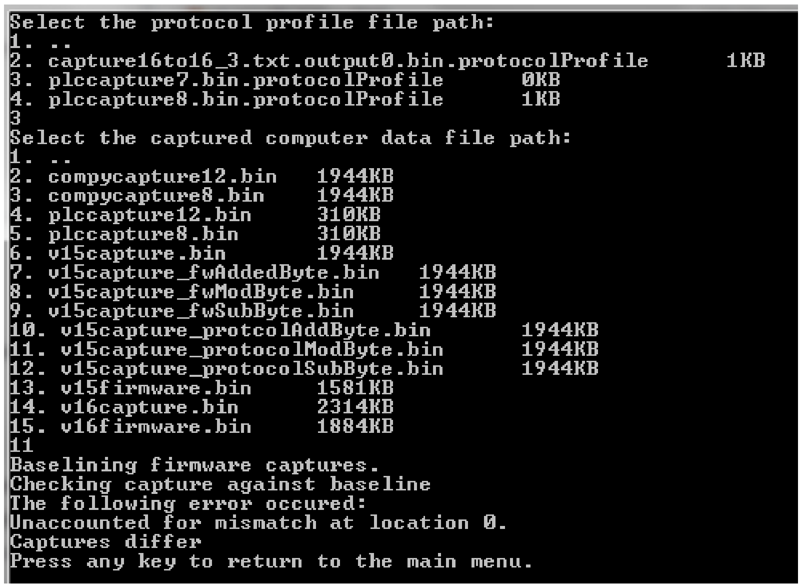
```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
12
Baselining firmware captures.
Checking capture against baseline
The following error occured:
Second data file does not contain enough bytes.
Captures differ
Press any key to return to the main menu.
```

Figure D.7: Byte removed from protocol data verification test.



```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
8
Baselining firmware captures.
Checking capture against baseline
The following error occured:
Unaccounted for mismatch at location 0.
Captures differ
Press any key to return to the main menu.
```

Figure D.8: Byte modified in firmware verification test.

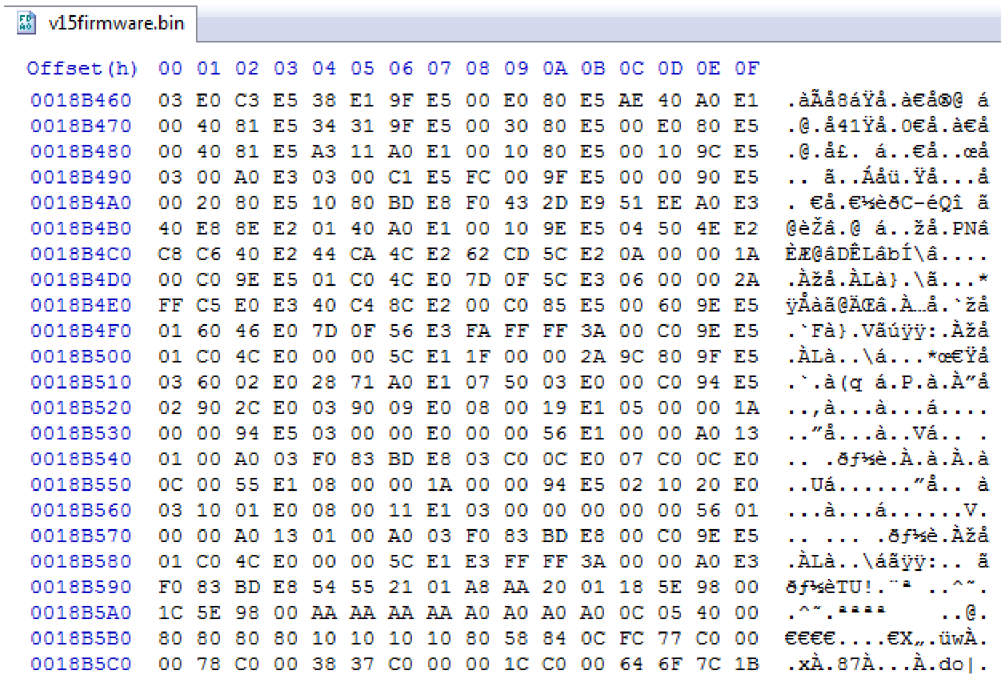


```
Select the protocol profile file path:
1. ..
2. capture16to16_3.txt.output0.bin.protocolProfile      1KB
3. plccapture7.bin.protocolProfile      0KB
4. plccapture8.bin.protocolProfile      1KB
3
Select the captured computer data file path:
1. ..
2. compycapture12.bin   1944KB
3. compycapture8.bin    1944KB
4. plccapture12.bin     310KB
5. plccapture8.bin      310KB
6. v15capture.bin       1944KB
7. v15capture_fwAddedByte.bin   1944KB
8. v15capture_fwModByte.bin     1944KB
9. v15capture_fwSubByte.bin     1944KB
10. v15capture_protcolAddByte.bin       1944KB
11. v15capture_protocolModByte.bin      1944KB
12. v15capture_protocolSubByte.bin      1944KB
13. v15firmware.bin     1581KB
14. v16capture.bin      2314KB
15. v16firmware.bin     1884KB
11
Baselining firmware captures.
Checking capture against baseline
The following error occured:
Unaccounted for mismatch at location 0.
Captures differ
Press any key to return to the main menu.
```

Figure D.9: Byte modified in protocol data verification test.

Appendix E: Test Case Modifications

The following screenshots display firmware modification test case data.

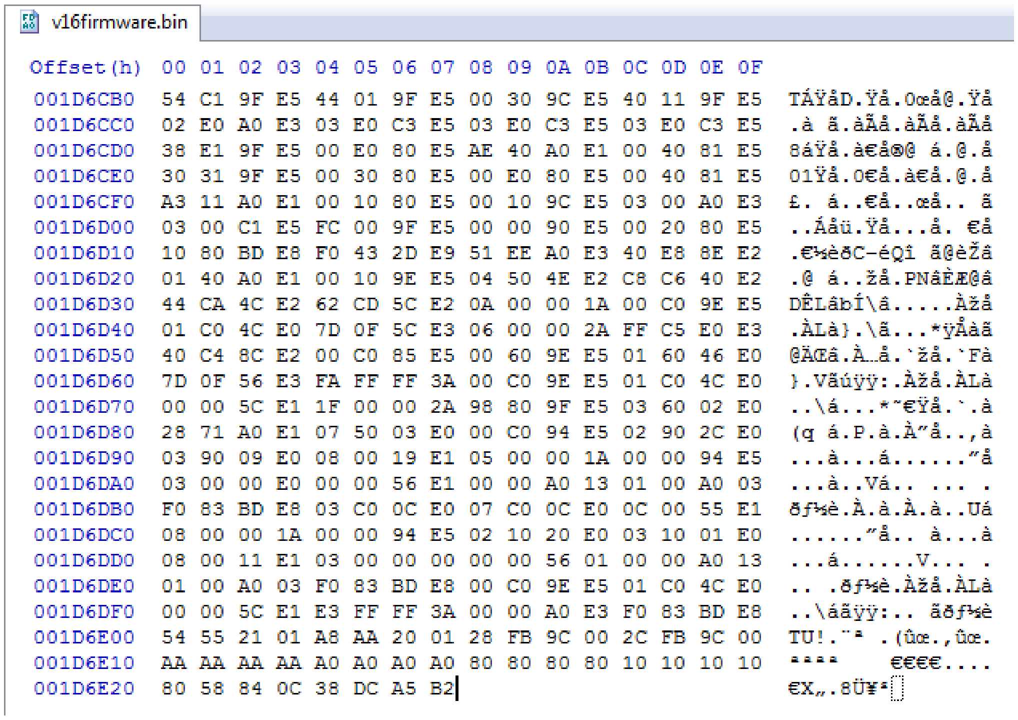


v15firmware.bin

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0018B460  03 E0 C3 E5 38 E1 9F E5 00 E0 80 E5 AE 40 A0 E1  .àÃå8áŸå.à€å®@ á
0018B470  00 40 81 E5 34 31 9F E5 00 30 80 E5 00 E0 80 E5  .@.å41Ÿå.0€å.à€å
0018B480  00 40 81 E5 A3 11 A0 E1 00 10 80 E5 00 10 9C E5  .@.å£. á..€å..œå
0018B490  03 00 A0 E3 03 00 C1 E5 FC 00 9F E5 00 00 90 E5  .. ã..Áåü.Ÿå...å
0018B4A0  00 20 80 E5 10 80 BD E8 F0 43 2D E9 51 EE A0 E3  . €å.€½èðC-éQî ã
0018B4B0  40 E8 8E E2 01 40 A0 E1 00 10 9E E5 04 50 4E E2  @èŽâ.@ á..žå.PNâ
0018B4C0  C8 C6 40 E2 44 CA 4C E2 62 CD 5C E2 0A 00 00 1A  ÈÆ@âDÊLâbÍ\â....
0018B4D0  00 C0 9E E5 01 C0 4C E0 7D 0F 5C E3 06 00 00 2A  .Àžå.ÀLà}.\ã...*
0018B4E0  FF C5 E0 E3 40 C4 8C E2 00 C0 85 E5 00 60 9E E5  ÿÅàã@ÄŒâ.À…å.`žå
0018B4F0  01 60 46 E0 7D 0F 56 E3 FA FF FF 3A 00 C0 9E E5  .`Fà}.Vãúÿÿ:.Àžå
0018B500  01 C0 4C E0 00 00 5C E1 1F 00 00 2A 9C 80 9F E5  .ÀLà..\á...*œ€Ÿå
0018B510  03 60 02 E0 28 71 A0 E1 07 50 03 E0 00 C0 94 E5  .`.à(q á.P.à.À”å
0018B520  02 90 2C E0 03 90 09 E0 08 00 19 E1 05 00 00 1A  ..,à...à...á....
0018B530  00 00 94 E5 03 00 00 E0 00 00 56 E1 00 00 A0 13  ..”å...à..Vá.. .
0018B540  01 00 A0 03 F0 83 BD E8 03 C0 0C E0 07 C0 0C E0  .. .ðƒ½è.À.à.À.à
0018B550  0C 00 55 E1 08 00 00 1A 00 00 94 E5 02 10 20 E0  ..Uá......”å.. à
0018B560  03 10 01 E0 08 00 11 E1 03 00 00 00 00 00 56 01  ...à...á......V.
0018B570  00 00 A0 13 01 00 A0 03 F0 83 BD E8 00 C0 9E E5  .. ... .ðƒ½è.Àžå
0018B580  01 C0 4C E0 00 00 5C E1 E3 FF FF 3A 00 00 A0 E3  .ÀLà..\áãÿÿ:.. ã
0018B590  F0 83 BD E8 54 55 21 01 A8 AA 20 01 18 5E 98 00  ðƒ½èTU!.¨ª ..^˜.
0018B5A0  1C 5E 98 00 AA AA AA AA A0 A0 A0 A0 0C 05 40 00  .^˜.ªªªª    ..@.
0018B5B0  80 80 80 80 10 10 10 10 80 58 84 0C FC 77 C0 00  €€€€....€X„.üwÀ.
0018B5C0  00 78 C0 00 38 37 C0 00 00 1C C0 00 64 6F 7C 1B  .xÀ.87À...À.do|.
```

Figure E.1: Version 15 firmware screenshot.

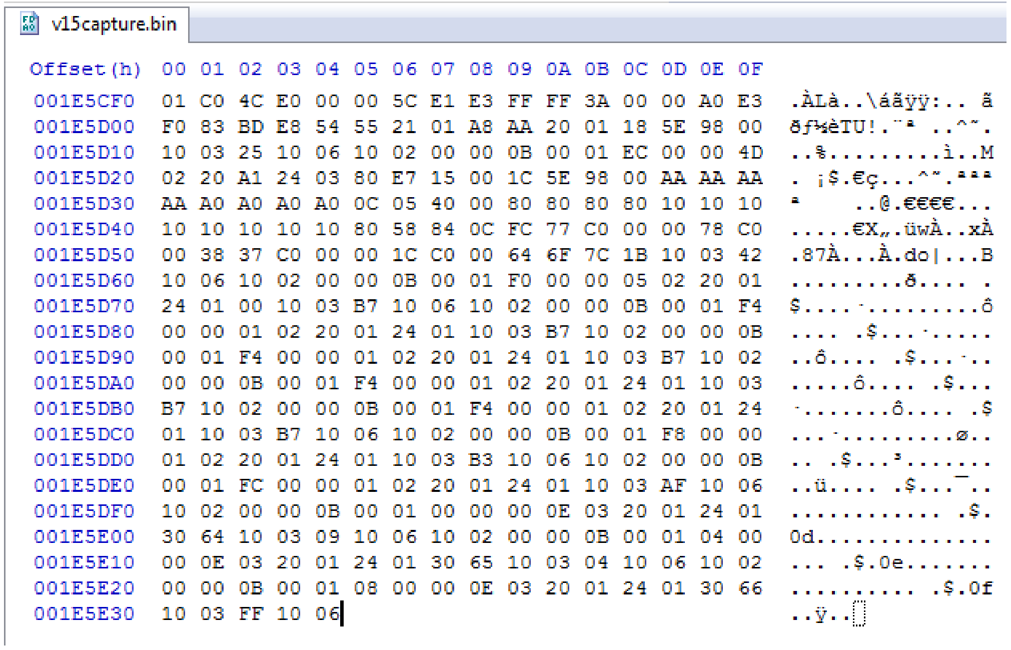


```
v16firmware.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001D6CB0  54 C1 9F E5 44 01 9F E5 00 30 9C E5 40 11 9F E5  TÁŸåD.Ÿå.0œå@.Ÿå
001D6CC0  02 E0 A0 E3 03 E0 C3 E5 03 E0 C3 E5 03 E0 C3 E5  .à ã.àÃå.àÃå.àÃå
001D6CD0  38 E1 9F E5 00 E0 80 E5 AE 40 A0 E1 00 40 81 E5  8áŸå.à€å®@ á.@.å
001D6CE0  30 31 9F E5 00 30 80 E5 00 E0 80 E5 00 40 81 E5  01Ÿå.0€å.à€å.@.å
001D6CF0  A3 11 A0 E1 00 10 80 E5 00 10 9C E5 03 00 A0 E3  £. á..€å..œå.. ã
001D6D00  03 00 C1 E5 FC 00 9F E5 00 00 90 E5 00 20 80 E5  ..Áåü.Ÿå...å. €å
001D6D10  10 80 BD E8 F0 43 2D E9 51 EE A0 E3 40 E8 8E E2  .€½èðC-éQî ã@èŽâ
001D6D20  01 40 A0 E1 00 10 9E E5 04 50 4E E2 C8 C6 40 E2  .@ á..žå.PNâÈÆ@â
001D6D30  44 CA 4C E2 62 CD 5C E2 0A 00 00 1A 00 C0 9E E5  DÊLâbÍ\â.....Àžå
001D6D40  01 C0 4C E0 7D 0F 5C E3 06 00 00 2A FF C5 E0 E3  .ÀLà}.\ã...*ÿÅàã
001D6D50  40 C4 8C E2 00 C0 85 E5 00 60 9E E5 01 60 46 E0  @ÄŒâ.À…å.`žå.`Fà
001D6D60  7D 0F 56 E3 FA FF FF 3A 00 C0 9E E5 01 C0 4C E0  }.Vãúÿÿ:.Àžå.ÀLà
001D6D70  00 00 5C E1 1F 00 00 2A 98 80 9F E5 03 60 02 E0  ..\á...*˜€Ÿå.`.à
001D6D80  28 71 A0 E1 07 50 03 E0 00 C0 94 E5 02 90 2C E0  (q á.P.à.À”å..,à
001D6D90  03 90 09 E0 08 00 19 E1 05 00 00 1A 00 00 94 E5  ...à...á......”å
001D6DA0  03 00 00 E0 00 00 56 E1 00 00 A0 13 01 00 A0 03  ...à..Vá.. ... .
001D6DB0  F0 83 BD E8 03 C0 0C E0 07 C0 0C E0 0C 00 55 E1  ðƒ½è.À.à.À.à..Uá
001D6DC0  08 00 00 1A 00 00 94 E5 02 10 20 E0 03 10 01 E0  ......”å.. à...à
001D6DD0  08 00 11 E1 03 00 00 00 00 00 56 01 00 00 A0 13  ...á......V... .
001D6DE0  01 00 A0 03 F0 83 BD E8 00 C0 9E E5 01 C0 4C E0  .. .ðƒ½è.Àžå.ÀLà
001D6DF0  00 00 5C E1 E3 FF FF 3A 00 00 A0 E3 F0 83 BD E8  ..\áãÿÿ:.. ãðƒ½è
001D6E00  54 55 21 01 A8 AA 20 01 28 FB 9C 00 2C FB 9C 00  TU!.¨ª .(ûœ.,ûœ.
001D6E10  AA AA AA AA A0 A0 A0 A0 80 80 80 80 10 10 10 10  ªªªª    €€€€....
001D6E20  80 58 84 0C 38 DC A5 B2                          €X„.8Ü¥²
```

Figure E.2: Version 15 firmware screenshot.

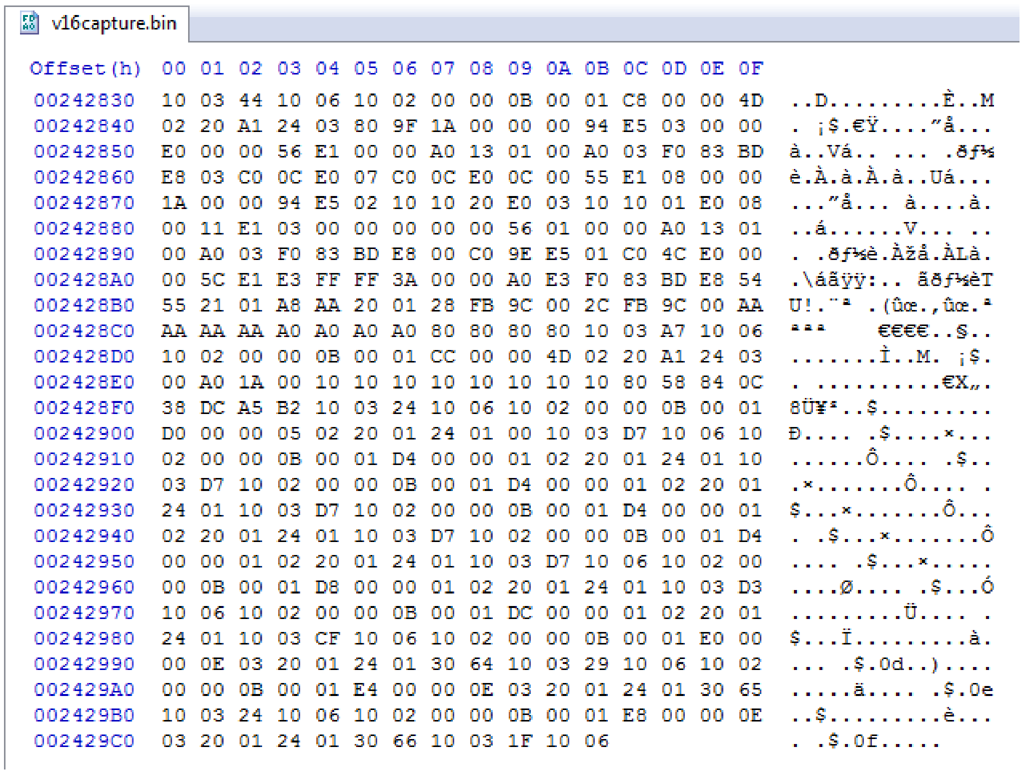


```
v15capture.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5CF0  01 C0 4C E0 00 00 5C E1 E3 FF FF 3A 00 00 A0 E3  .ÀLà..\áãÿÿ:.. ã
001E5D00  F0 83 BD E8 54 55 21 01 A8 AA 20 01 18 5E 98 00  ðƒ½èTU!.¨ª ..^˜.
001E5D10  10 03 25 10 06 10 02 00 00 0B 00 01 EC 00 00 4D  ..%.........ì..M
001E5D20  02 20 A1 24 03 80 E7 15 00 1C 5E 98 00 AA AA AA  . ¡$.€ç...^˜.ªªª
001E5D30  AA A0 A0 A0 A0 0C 05 40 00 80 80 80 80 10 10 10  ª    ..@.€€€€...
001E5D40  10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50  00 38 37 C0 00 00 1C C0 00 64 6F 7C 1B 10 03 42  .87À...À.do|...B
001E5D60  10 06 10 02 00 00 0B 00 01 F0 00 00 05 02 20 01  .........ð.... .
001E5D70  24 01 00 10 03 B7 10 06 10 02 00 00 0B 00 01 F4  $....·.........ô
001E5D80  00 00 01 02 20 01 24 01 10 03 B7 10 02 00 00 0B  .... .$...·.....
001E5D90  00 01 F4 00 00 01 02 20 01 24 01 10 03 B7 10 02  ..ô.... .$...·..
001E5DA0  00 00 0B 00 01 F4 00 00 01 02 20 01 24 01 10 03  .....ô.... .$...
001E5DB0  B7 10 02 00 00 0B 00 01 F4 00 00 01 02 20 01 24  ·.......ô.... .$
001E5DC0  01 10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00 00  ...·.........ø..
001E5DD0  01 02 20 01 24 01 10 03 B3 10 06 10 02 00 00 0B  .. .$...³.......
001E5DE0  00 01 FC 00 00 01 02 20 01 24 01 10 03 AF 10 06  ..ü.... .$...¯..
001E5DF0  10 02 00 00 0B 00 01 00 00 00 0E 03 20 01 24 01  ............ .$.
001E5E00  30 64 10 03 09 10 06 10 02 00 00 0B 00 01 04 00  0d..............
001E5E10  00 0E 03 20 01 24 01 30 65 10 03 04 10 06 10 02  ... .$.0e.......
001E5E20  00 00 0B 00 01 08 00 00 0E 03 20 01 24 01 30 66  .......... .$.0f
001E5E30  10 03 FF 10 06                                   ..ÿ..
```

Figure E.3: Version 15 capture screenshot.

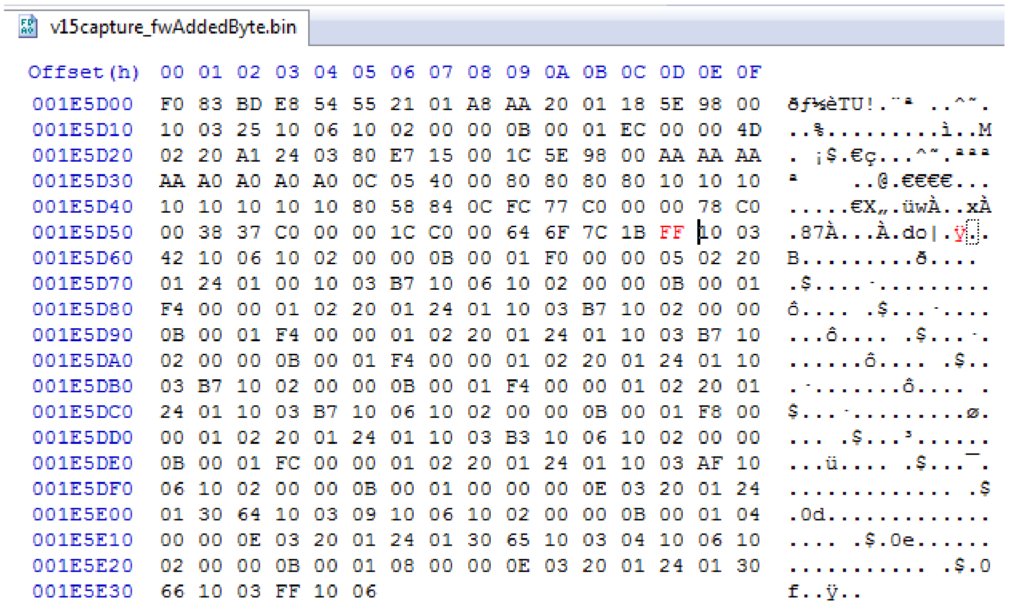


```
v16capture.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00242830  10 03 44 10 06 10 02 00 00 0B 00 01 C8 00 00 4D  ..D.........È..M
00242840  02 20 A1 24 03 80 9F 1A 00 00 00 94 E5 03 00 00  . ¡$.€Ÿ....”å...
00242850  E0 00 00 56 E1 00 00 A0 13 01 00 A0 03 F0 83 BD  à..Vá.. ... .ðƒ½
00242860  E8 03 C0 0C E0 07 C0 0C E0 0C 00 55 E1 08 00 00  è.À.à.À.à..Uá...
00242870  1A 00 00 94 E5 02 10 10 20 E0 03 10 10 01 E0 08  ...”å... à....à.
00242880  00 11 E1 03 00 00 00 00 00 56 01 00 00 A0 13 01  ..á......V... ..
00242890  00 A0 03 F0 83 BD E8 00 C0 9E E5 01 C0 4C E0 00  . .ðƒ½è.Àžå.ÀLà.
002428A0  00 5C E1 E3 FF FF 3A 00 00 A0 E3 F0 83 BD E8 54  .\áãÿÿ:.. ãðƒ½èT
002428B0  55 21 01 A8 AA 20 01 28 FB 9C 00 2C FB 9C 00 AA  U!.¨ª .(ûœ.,ûœ.ª
002428C0  AA AA AA A0 A0 A0 A0 80 80 80 80 10 03 A7 10 06  ªªª    €€€€..§..
002428D0  10 02 00 00 0B 00 01 CC 00 00 4D 02 20 A1 24 03  .......Ì..M. ¡$.
002428E0  00 A0 1A 00 10 10 10 10 10 10 10 10 80 58 84 0C  . ..........€X„.
002428F0  38 DC A5 B2 10 03 24 10 06 10 02 00 00 0B 00 01  8Ü¥²..$.........
00242900  D0 00 00 05 02 20 01 24 01 00 10 03 D7 10 06 10  Ð.... .$....×...
00242910  02 00 00 0B 00 01 D4 00 00 01 02 20 01 24 01 10  ......Ô.... .$..
00242920  03 D7 10 02 00 00 0B 00 01 D4 00 00 01 02 20 01  .×.......Ô.... .
00242930  24 01 10 03 D7 10 02 00 00 0B 00 01 D4 00 00 01  $...×.......Ô...
00242940  02 20 01 24 01 10 03 D7 10 02 00 00 0B 00 01 D4  . .$...×.......Ô
00242950  00 00 01 02 20 01 24 01 10 03 D7 10 06 10 02 00  .... .$...×.....
00242960  00 0B 00 01 D8 00 00 01 02 20 01 24 01 10 03 D3  ....Ø.... .$...Ó
00242970  10 06 10 02 00 00 0B 00 01 DC 00 00 01 02 20 01  .........Ü.... .
00242980  24 01 10 03 CF 10 06 10 02 00 00 0B 00 01 E0 00  $...Ï.........à.
00242990  00 0E 03 20 01 24 01 30 64 10 03 29 10 06 10 02  ... .$.0d..)....
002429A0  00 00 0B 00 01 E4 00 00 0E 03 20 01 24 01 30 65  .....ä.... .$.0e
002429B0  10 03 24 10 06 10 02 00 00 0B 00 01 E8 00 00 0E  ..$.........è...
002429C0  03 20 01 24 01 30 66 10 03 1F 10 06              . .$.0f.....
```

Figure E.4: Version 16 capture screenshot.

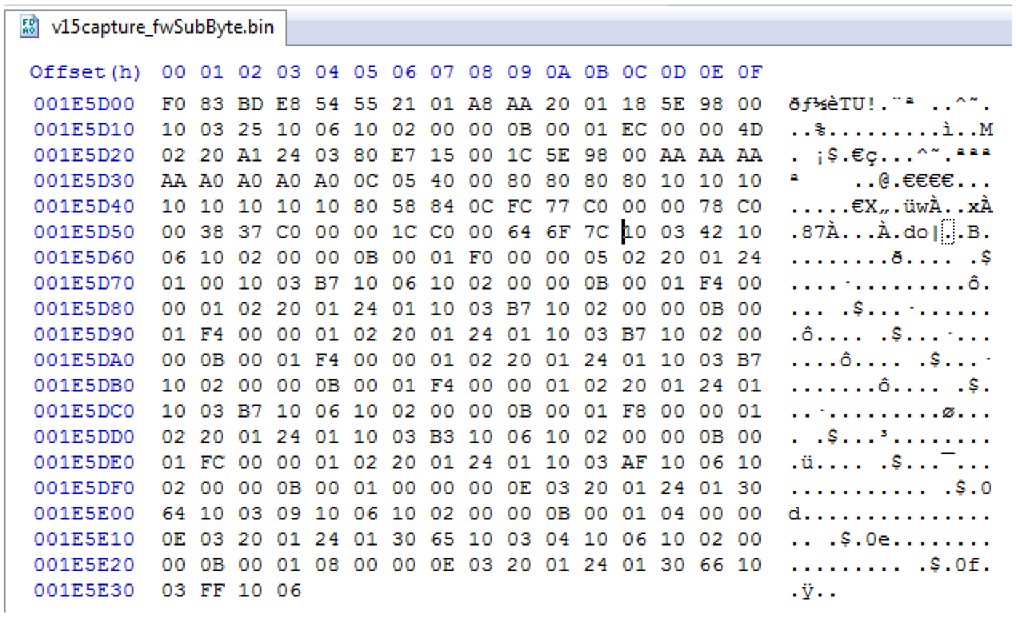


```
v15capture_fwAddedByte.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5D00  F0 83 BD E8 54 55 21 01 A8 AA 20 01 18 5E 98 00  ðƒ½èTU!.¨ª ..^˜.
001E5D10  10 03 25 10 06 10 02 00 00 0B 00 01 EC 00 00 4D  ..%.........ì..M
001E5D20  02 20 A1 24 03 80 E7 15 00 1C 5E 98 00 AA AA AA  . ¡$.€ç...^˜.ªªª
001E5D30  AA A0 A0 A0 A0 0C 05 40 00 80 80 80 80 10 10 10  ª    ..@.€€€€...
001E5D40  10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50  00 38 37 C0 00 00 1C C0 00 64 6F 7C 1B FF 10 03  .87À...À.do|.ÿ..
001E5D60  42 10 06 10 02 00 00 0B 00 01 F0 00 00 05 02 20  B..........ð....
001E5D70  01 24 01 00 10 03 B7 10 06 10 02 00 00 0B 00 01  .$....·.........
001E5D80  F4 00 00 01 02 20 01 24 01 10 03 B7 10 02 00 00  ô.... .$...·....
001E5D90  0B 00 01 F4 00 00 01 02 20 01 24 01 10 03 B7 10  ...ô.... .$...·.
001E5DA0  02 00 00 0B 00 01 F4 00 00 01 02 20 01 24 01 10  ......ô.... .$..
001E5DB0  03 B7 10 02 00 00 0B 00 01 F4 00 00 01 02 20 01  .·.......ô.... .
001E5DC0  24 01 10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00  $...·.........ø.
001E5DD0  00 01 02 20 01 24 01 10 03 B3 10 06 10 02 00 00  ... .$...³......
001E5DE0  0B 00 01 FC 00 00 01 02 20 01 24 01 10 03 AF 10  ...ü.... .$...¯.
001E5DF0  06 10 02 00 00 0B 00 01 00 00 00 0E 03 20 01 24  ............. .$
001E5E00  01 30 64 10 03 09 10 06 10 02 00 00 0B 00 01 04  .0d.............
001E5E10  00 00 0E 03 20 01 24 01 30 65 10 03 04 10 06 10  .... .$.0e......
001E5E20  02 00 00 0B 00 01 08 00 00 0E 03 20 01 24 01 30  ........... .$.0
001E5E30  66 10 03 FF 10 06                                f..ÿ..
```

Figure E.5: Byte added to firmware data.

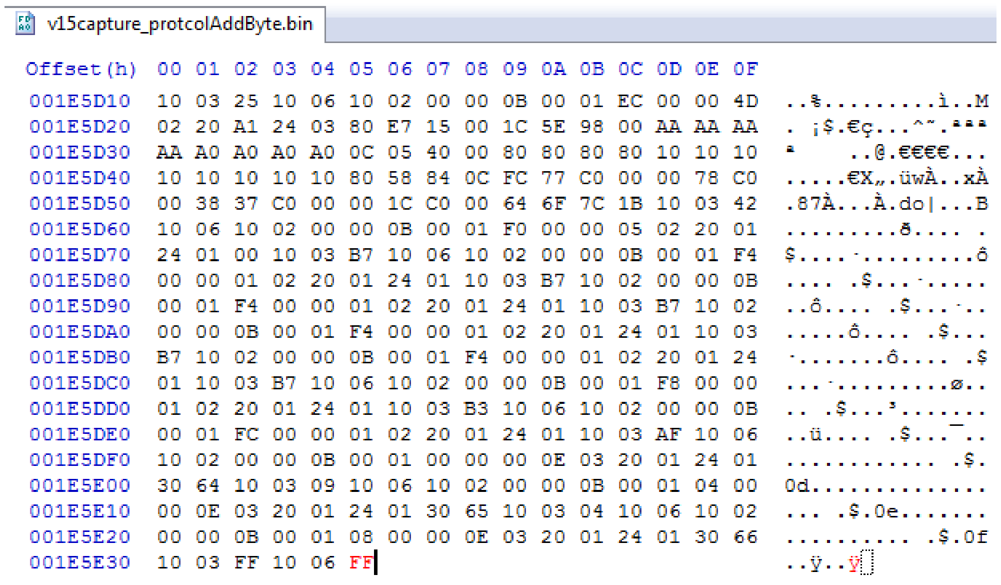


```
v15capture_fwSubByte.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5D00  F0 83 BD E8 54 55 21 01 A8 AA 20 01 18 5E 98 00  ðƒ½èTU!.¨ª ..^˜.
001E5D10  10 03 25 10 06 10 02 00 00 0B 00 01 EC 00 00 4D  ..%.........ì..M
001E5D20  02 20 A1 24 03 80 E7 15 00 1C 5E 98 00 AA AA AA  . ¡$.€ç...^˜.ªªª
001E5D30  AA A0 A0 A0 A0 0C 05 40 00 80 80 80 80 10 10 10  ª    ..@.€€€€...
001E5D40  10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50  00 38 37 C0 00 00 1C C0 00 64 6F 7C 10 03 42 10  .87À...À.do|..B.
001E5D60  06 10 02 00 00 0B 00 01 F0 00 00 05 02 20 01 24  ........ð.... .$
001E5D70  01 00 10 03 B7 10 06 10 02 00 00 0B 00 01 F4 00  ....·.........ô.
001E5D80  00 01 02 20 01 24 01 10 03 B7 10 02 00 00 0B 00  ... .$...·......
001E5D90  01 F4 00 00 01 02 20 01 24 01 10 03 B7 10 02 00  .ô.... .$...·...
001E5DA0  00 0B 00 01 F4 00 00 01 02 20 01 24 01 10 03 B7  ....ô.... .$...·
001E5DB0  10 02 00 00 0B 00 01 F4 00 00 01 02 20 01 24 01  .......ô.... .$.
001E5DC0  10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00 00 01  ..·.........ø...
001E5DD0  02 20 01 24 01 10 03 B3 10 06 10 02 00 00 0B 00  . .$...³........
001E5DE0  01 FC 00 00 01 02 20 01 24 01 10 03 AF 10 06 10  .ü.... .$...¯...
001E5DF0  02 00 00 0B 00 01 00 00 00 0E 03 20 01 24 01 30  ........... .$.0
001E5E00  64 10 03 09 10 06 10 02 00 00 0B 00 01 04 00 00  d...............
001E5E10  0E 03 20 01 24 01 30 65 10 03 04 10 06 10 02 00  .. .$.0e........
001E5E20  00 0B 00 01 08 00 00 0E 03 20 01 24 01 30 66 10  ......... .$.0f.
001E5E30  03 FF 10 06                                      .ÿ..
```

Figure E.6: Byte removed from firmware data.

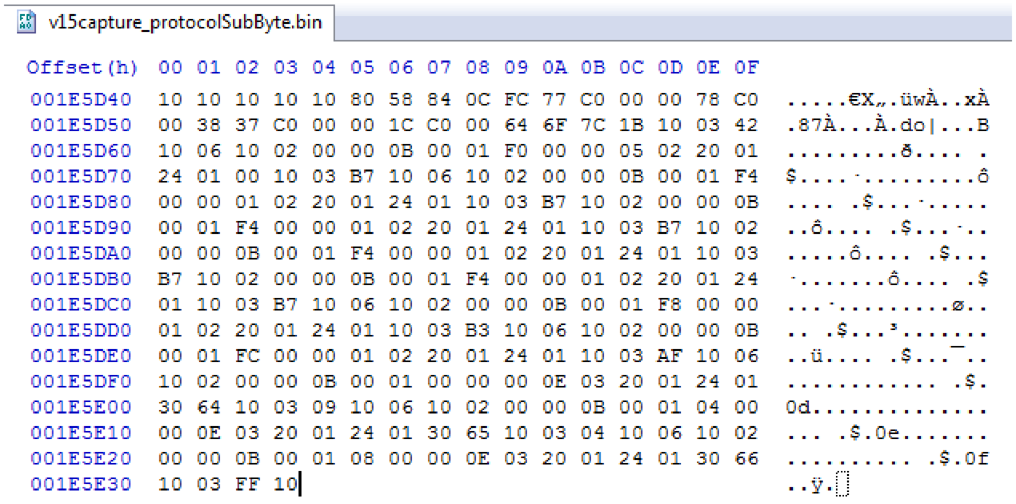


```
v15capture_protcolAddByte.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5D10  10 03 25 10 06 10 02 00 00 0B 00 01 EC 00 00 4D  ..%.........ì..M
001E5D20  02 20 A1 24 03 80 E7 15 00 1C 5E 98 00 AA AA AA  . ¡$.€ç...^˜.ªªª
001E5D30  AA A0 A0 A0 A0 0C 05 40 00 80 80 80 80 10 10 10  ª    ..@.€€€€...
001E5D40  10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50  00 38 37 C0 00 00 1C C0 00 64 6F 7C 1B 10 03 42  .87À...À.do|...B
001E5D60  10 06 10 02 00 00 0B 00 01 F0 00 00 05 02 20 01  .........ð.... .
001E5D70  24 01 00 10 03 B7 10 06 10 02 00 00 0B 00 01 F4  $....·.........ô
001E5D80  00 00 01 02 20 01 24 01 10 03 B7 10 02 00 00 0B  .... .$...·.....
001E5D90  00 01 F4 00 00 01 02 20 01 24 01 10 03 B7 10 02  ..ô.... .$...·..
001E5DA0  00 00 0B 00 01 F4 00 00 01 02 20 01 24 01 10 03  .....ô.... .$...
001E5DB0  B7 10 02 00 00 0B 00 01 F4 00 00 01 02 20 01 24  ·.......ô.... .$
001E5DC0  01 10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00 00  ...·.........ø..
001E5DD0  01 02 20 01 24 01 10 03 B3 10 06 10 02 00 00 0B  .. .$...³.......
001E5DE0  00 01 FC 00 00 01 02 20 01 24 01 10 03 AF 10 06  ..ü.... .$...¯..
001E5DF0  10 02 00 00 0B 00 01 00 00 00 0E 03 20 01 24 01  ............ .$.
001E5E00  30 64 10 03 09 10 06 10 02 00 00 0B 00 01 04 00  0d..............
001E5E10  00 0E 03 20 01 24 01 30 65 10 03 04 10 06 10 02  ... .$.0e.......
001E5E20  00 00 0B 00 01 08 00 00 0E 03 20 01 24 01 30 66  .......... .$.0f
001E5E30  10 03 FF 10 06 FF                                ..ÿ..ÿ
```

Figure E.7: Byte added to protocol data.

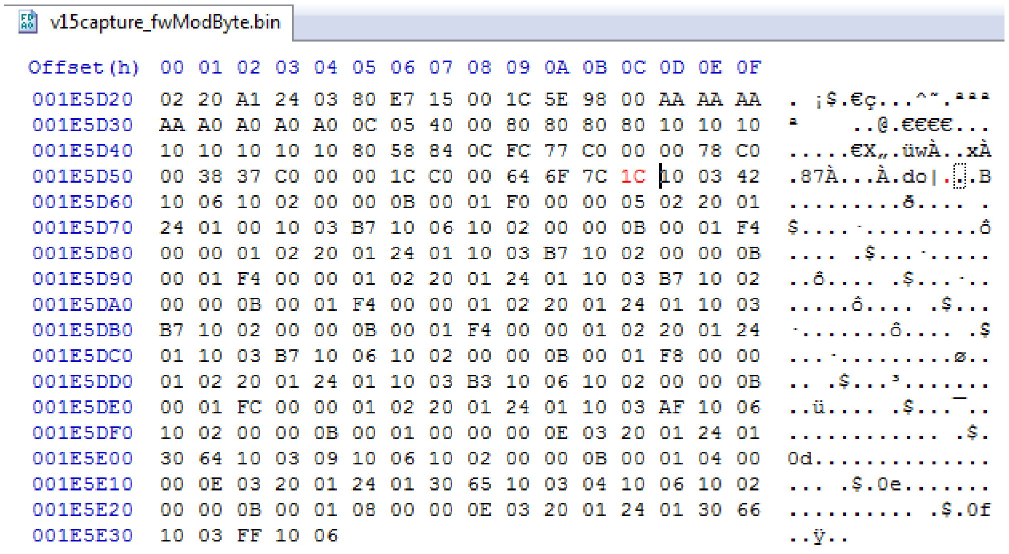


```
v15capture_protocolSubByte.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5D40  10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50  00 38 37 C0 00 00 1C C0 00 64 6F 7C 1B 10 03 42  .87À...À.do|...B
001E5D60  10 06 10 02 00 00 0B 00 01 F0 00 00 05 02 20 01  .........ð.... .
001E5D70  24 01 00 10 03 B7 10 06 10 02 00 00 0B 00 01 F4  $....·.........ô
001E5D80  00 00 01 02 20 01 24 01 10 03 B7 10 02 00 00 0B  .... .$...·.....
001E5D90  00 01 F4 00 00 01 02 20 01 24 01 10 03 B7 10 02  ..ô.... .$...·..
001E5DA0  00 00 0B 00 01 F4 00 00 01 02 20 01 24 01 10 03  .....ô.... .$...
001E5DB0  B7 10 02 00 00 0B 00 01 F4 00 00 01 02 20 01 24  ·.......ô.... .$
001E5DC0  01 10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00 00  ...·.........ø..
001E5DD0  01 02 20 01 24 01 10 03 B3 10 06 10 02 00 00 0B  .. .$...³.......
001E5DE0  00 01 FC 00 00 01 02 20 01 24 01 10 03 AF 10 06  ..ü.... .$...¯..
001E5DF0  10 02 00 00 0B 00 01 00 00 00 0E 03 20 01 24 01  ............ .$.
001E5E00  30 64 10 03 09 10 06 10 02 00 00 0B 00 01 04 00  0d..............
001E5E10  00 0E 03 20 01 24 01 30 65 10 03 04 10 06 10 02  ... .$.0e.......
001E5E20  00 00 0B 00 01 08 00 00 0E 03 20 01 24 01 30 66  .......... .$.0f
001E5E30  10 03 FF 10                                      ..ÿ.
```

Figure E.8: Byte removed from protocol data.

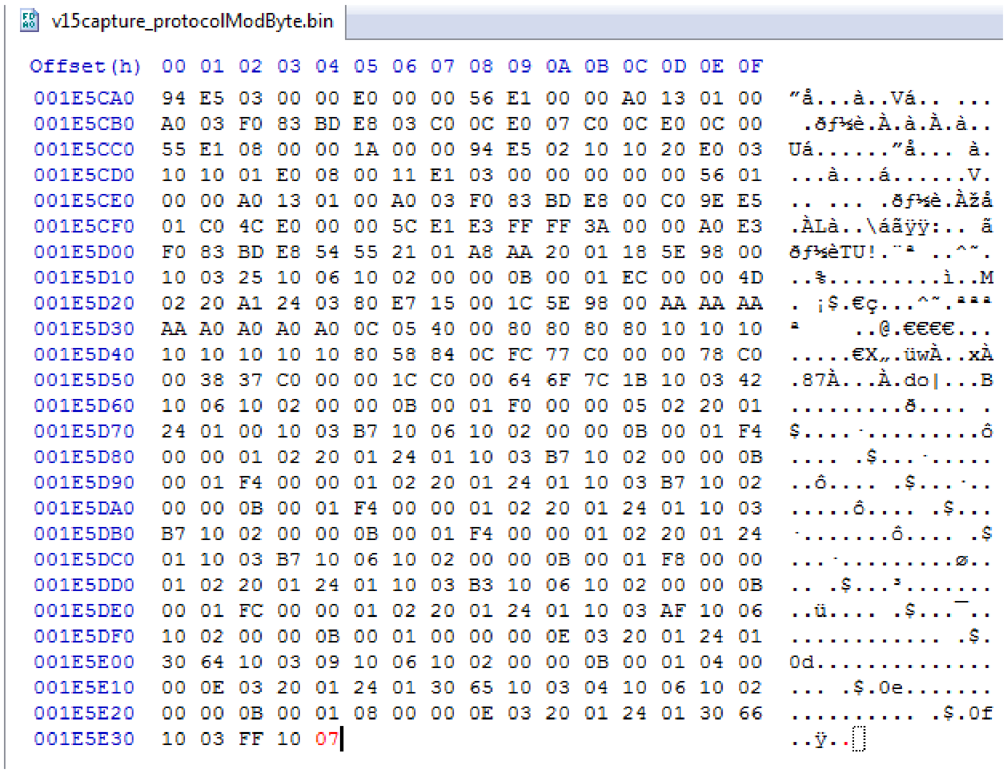


```
v15capture_fwModByte.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5D20  02 20 A1 24 03 80 E7 15 00 1C 5E 98 00 AA AA AA  . ¡$.€ç...^˜.ªªª
001E5D30  AA A0 A0 A0 A0 0C 05 40 00 80 80 80 80 10 10 10  ª    ..@.€€€€...
001E5D40  10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50  00 38 37 C0 00 00 1C C0 00 64 6F 7C 1C 10 03 42  .87À...À.do|...B
001E5D60  10 06 10 02 00 00 0B 00 01 F0 00 00 05 02 20 01  .........ð.... .
001E5D70  24 01 00 10 03 B7 10 06 10 02 00 00 0B 00 01 F4  $....·.........ô
001E5D80  00 00 01 02 20 01 24 01 10 03 B7 10 02 00 00 0B  .... .$...·.....
001E5D90  00 01 F4 00 00 01 02 20 01 24 01 10 03 B7 10 02  ..ô.... .$...·..
001E5DA0  00 00 0B 00 01 F4 00 00 01 02 20 01 24 01 10 03  .....ô.... .$...
001E5DB0  B7 10 02 00 00 0B 00 01 F4 00 00 01 02 20 01 24  ·.......ô.... .$
001E5DC0  01 10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00 00  ...·.........ø..
001E5DD0  01 02 20 01 24 01 10 03 B3 10 06 10 02 00 00 0B  .. .$...³.......
001E5DE0  00 01 FC 00 00 01 02 20 01 24 01 10 03 AF 10 06  ..ü.... .$...¯..
001E5DF0  10 02 00 00 0B 00 01 00 00 00 0E 03 20 01 24 01  ............ .$.
001E5E00  30 64 10 03 09 10 06 10 02 00 00 0B 00 01 04 00  0d..............
001E5E10  00 0E 03 20 01 24 01 30 65 10 03 04 10 06 10 02  ... .$.0e.......
001E5E20  00 00 0B 00 01 08 00 00 0E 03 20 01 24 01 30 66  .......... .$.0f
001E5E30  10 03 FF 10 06                                   ..ÿ..
```

Figure E.9: Byte modified in firmware data.



```
v15capture_protocolModByte.bin

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

001E5CA0   94 E5 03 00 00 E0 00 00 56 E1 00 00 A0 13 01 00  ”å...à..Vá.. ...
001E5CB0   A0 03 F0 83 BD E8 03 C0 0C E0 07 C0 0C E0 0C 00   .ðƒ½è.À.à.À.à..
001E5CC0   55 E1 08 00 00 1A 00 00 94 E5 02 10 10 20 E0 03  Uá......”å... à.
001E5CD0   10 10 01 E0 08 00 11 E1 03 00 00 00 00 00 56 01  ...à...á......V.
001E5CE0   00 00 A0 13 01 00 A0 03 F0 83 BD E8 00 C0 9E E5  .. ... .ðƒ½è.Àžå
001E5CF0   01 C0 4C E0 00 00 5C E1 E3 FF FF 3A 00 00 A0 E3  .ÀLà..\áãÿÿ:.. ã
001E5D00   F0 83 BD E8 54 55 21 01 A8 AA 20 01 18 5E 98 00  ðƒ½èTU!.¨ª ..^˜.
001E5D10   10 03 25 10 06 10 02 00 00 0B 00 01 EC 00 00 4D  ..%.........ì..M
001E5D20   02 20 A1 24 03 80 E7 15 00 1C 5E 98 00 AA AA AA  . ¡$.€ç...^˜.ªªª
001E5D30   AA A0 A0 A0 A0 0C 05 40 00 80 80 80 80 10 10 10  ª    ..@.€€€€...
001E5D40   10 10 10 10 10 80 58 84 0C FC 77 C0 00 00 78 C0  .....€X„.üwÀ..xÀ
001E5D50   00 38 37 C0 00 00 1C C0 00 64 6F 7C 1B 10 03 42  .87À...À.do|...B
001E5D60   10 06 10 02 00 00 0B 00 01 F0 00 00 05 02 20 01  .........ð.... .
001E5D70   24 01 00 10 03 B7 10 06 10 02 00 00 0B 00 01 F4  $....·.........ô
001E5D80   00 00 01 02 20 01 24 01 10 03 B7 10 02 00 00 0B  .... .$...·.....
001E5D90   00 01 F4 00 00 01 02 20 01 24 01 10 03 B7 10 02  ..ô.... .$...·..
001E5DA0   00 00 0B 00 01 F4 00 00 01 02 20 01 24 01 10 03  .....ô.... .$...
001E5DB0   B7 10 02 00 00 0B 00 01 F4 00 00 01 02 20 01 24  ·.......ô.... .$
001E5DC0   01 10 03 B7 10 06 10 02 00 00 0B 00 01 F8 00 00  ...·.........ø..
001E5DD0   01 02 20 01 24 01 10 03 B3 10 06 10 02 00 00 0B  .. .$...³.......
001E5DE0   00 01 FC 00 00 01 02 20 01 24 01 10 03 AF 10 06  ..ü.... .$...¯..
001E5DF0   10 02 00 00 0B 00 01 00 00 00 0E 03 20 01 24 01  ............ .$.
001E5E00   30 64 10 03 09 10 06 10 02 00 00 0B 00 01 04 00  0d..............
001E5E10   00 0E 03 20 01 24 01 30 65 10 03 04 10 06 10 02  ... .$.0e.......
001E5E20   00 00 0B 00 01 08 00 00 0E 03 20 01 24 01 30 66  .......... .$.0f
001E5E30   10 03 FF 10 07                                   ..ÿ..
```

Figure E.10: Byte modified in protocol data.

Bibliography

[1] Stuart A. Boyer. *SCADA: Supervisory Control and Data Acquisition*. ISA, 3rd edition, 06 2004.

[2] Department of Homeland Security. National infrastructure protection plan. Technical report, Department of Homeland Security, 2009.

[3] IEEE. IEEE standard for substation intelligent electronic devices (IEDs) cyber security capabilities, 2008.

[4] NIST Computer Security Division. Managing information security risk: Organization, mission, and information system view. Special Publication 800-39, National Institute of Standards and Technology, U.S. Department of Commerce, 2011.

[5] Keith Stouffer, Joe Falco, and Karen Kent. Guide to supervisory control and data acquisition (SCADA) and industrial control systems security. Technical Report NIST-SP-800-82-2006, National Institute of Standards, 2011.

[6] Nicolas Falliere, Liam Murchu, and Eric Chien. W32.stuxnet dossier. Technical report, Symantec, 2011.

[7] William J. Lynn III. Defending a new doman: The pentagon's cyberstrategy. *Council on Foreign Affairs*, 2010.

[8] William T. Shaw. *Cybersecurity for SCADA Systems*. PennWell, Tulsa, OK, 2006.

[9] United States Government Accountability Office. Cybersecurity: Continued attention needed to protect our nation's critical infrastructure. Testimony GAO-11-865T, 2011.

[10] US-CERT. Control systems - standards and references, 2011.

[11] W. Bolton. *Programmable Logic Controllers*. Newnes, Woburn, MA, 2nd edition, 2000.

[12] John Crisp. *Introduction to microprocessors and microcontrollers*. Amsterdam; Elsevier/Newnes, 2004.

[13] Siemens. SIMATIC embedded automation S7 modular embedded controller. Technical Report A5E01716600-03, Siemens AG, 2009.

[14] Rockwell Software. Flash firmware updates, 2011.

[15] Moses D. Schwartz, John Mulder, Jason Trent, and William D. Atkins. Control system devices: Architectures and supply channels overview. Technical Report SAND2010-5183, Sandia National Laboratories, 2010.

[16] G. Gilchrist. Secure authentication for DNP3. In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pages 1–3, 2008.

[17] O. Pal, S. Saiwan, P. Jain, Z. Saquib, and D. Patel. Cryptographic key management for SCADA system: An architectural framework. In *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT '09. International Conference on*, pages 169–174, 2009.

[18] C. Basile, S. Di Carlo, and A. Scionti. FPGA-based remote-code integrity verification of programs in distributed embedded systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, PP(99):1–14, 2011.

[19] M. Jakobsson and K. A Johansson. Practical and secure software-based attestation. In *Lightweight Security & Privacy: Devices, Protocols and Applications (LightSec), 2011 Workshop on*, pages 1–9, 2011.

[20] Kyungsub Song, Dongwon Seo, Haemin Park, Heejo Lee, and A. Perrig. Omap: One-way memory attestation protocol for smart meters. In *Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on*, pages 111–118, 2011.

[21] T. Morris and K. Pavurapu. A retrofit network transaction data logger and intrusion detection system for transmission and distribution substations. In *Power and Energy (PECon), 2010 IEEE International Conference on*, pages 958–963, 2010.

[22] Nicolas Falliere. Exploring stuxnet's PLC infection process, 2010.

[23] Siobhan Gorman, August Cole, and Yochi Dreazen. Computer spies breach fighter-jet project. *The Wall Street Journal*, 2009.

[24] United States Government Accountability Office. Information security: Cyber threats and vulnerabilities place federal systems at risk. Congressional Testimony GAO-09-661T, 2009.

[25] Wei Gao, T. Morris, B. Reaves, and D. Richey. On SCADA control system command and response injection and intrusion detection. In *eCrime Researchers Summit (eCrime), 2010*, pages 1–9, 2010.

[26] Robert Turk. *Cyber Incidents Involving Control Systems*. US-CERT Control Systems Security Center; Idaho Falls, ID, 2005.

[27] Microsystems Technology Office. Integrity and reliability of integrated circuits (IRIS). Technical Report DARPA-BAA-10-33, DARPA, 2010.

[28] J. Stradley and D. Karraker. The electronic part supply chain and risks of counterfeit parts in defense applications. *Components and Packaging Technologies, IEEE Transactions on*, 29(3):703–705, 2006.

[29] Peter Huitsing, Rodrigo Chandia, Mauricio Papa, and Sujeet Shenoi. Attack taxonomies for the modbus protocols. *International Journal of Critical Infrastructure Protection*, 1(0):37–44, 12 2008.

[30] East S., Butts J., Papa M., and Shenoi S. A taxonomy of attacks on the DNP3 protocol. In C. Palmer & S. Shenoi, editor, *Critical Infrastructure Protection III*, page 67, 2009.

[31] I. N. Fovino, A. Carcano, and M. Masera. A secure and survivable architecture for SCADA systems. In *Dependability, 2009. DEPEND '09. Second International Conference on*, pages 34–39, 2009.

[32] T. Mander, F. Nabhani, Lin Wang, and R. Cheung. Data object based security for DNP3 over tcp/ip for increased utility commercial aspects security. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8, 2007.

[33] IEEE Power and Energy Society. IEEE standard for electric power systems communications – distributed network protocol (DNP3), 2010.

[34] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography: with Coding Theory*. Pearson Prentice Hall; Upper Saddle River, NJ, 2006.

[35] Ronald L. Rivest. RFC 1321 - the MD5 message-digest algorithm. Technical Report RFC 1321, IETF Tools, 1992.

[36] D. Dzung, M. Naedele, T. P. Von Hoff, and M. Crevatin. Security for industrial communication systems. *Proceedings of the IEEE*, 93(6):1152–1177, 2005.

[37] T. Feller, S. Malipatlolla, D. Meister, and S. A. Huss. TinyTPM: A lightweight module aimed to IP protection and trusted embedded platforms. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 6–11, 2011.

[38] A. Khan, M. K. Sharma, G. Ganesh, S. D. Dhodapkar, B. B. Biswas, and R. K. Patil. A cryptographic primitive based authentication scheme for run-time software of embedded systems. In *Reliability, Safety and Hazard (ICRESH), 2010 2nd International Conference on*, pages 500–504, 2010.

[39] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: software-based attestation for embedded devices. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 272–282, 2004.

[40] Vinay M. Igure, Sean A. Laughter, and Ronald D. Williams. Security issues in SCADA networks. *Computers & Security*, 25(7):498–506, 10 2006.

[41] T. AbuHmed, N. Nyamaa, and DaeHun Nyang. Software-based remote code attestation in wireless sensor network. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–8, 2009.

[42] D. K. Nilsson, Lei Sun, and T. Nakajima. A framework for self-verification of firmware updates over the air in vehicle ECUs. In *GLOBECOM Workshops, 2008 IEEE*, pages 1–5, 2008.

[43] Ce Meng, Yeping He, and Qian Zhang. Remote attestation for custom-built software. In *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC '09. International Conference on*, volume 2, pages 374–377, 2009.

[44] IEEE Power and Energy Society. IEEE recommended practice for microprocessor-based protection equipment firmware control, 2006.

[45] Defense Industry Daily. Secure semiconductors: Sensible, or sisyphean?, 2011.

[46] Dean Collins. Darpa "trust in IC's" effort. Technical report, Microsystems Technology Office, DARPA, 2007.

[47] F. E. McFadden and R. D. Arnold. Supply chain risk mitigation for it electronics. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pages 49–55, 2010.

[48] S. Adee. The hunt for the kill switch. *Spectrum, IEEE*, 45(5):34–39, 2008.

[49] M. Tehranipoor, H. Salmani, Xuehui Zhang, Xiaoxiao Wang, R. Karri, J. Rajendran, and K. Rosenfeld. Trustworthy hardware: Trojan detection and design-for-trust challenges. *Computer*, 44(7):66–74, 2011.

[50] Xiaoxiao Wang, M. Tehranipoor, and J. Plusquellic. Detecting malicious inclusions in secure hardware: Challenges and solutions. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 15–19, 2008.

[51] Department of Homeland Security. Strategy for securing control systems. Technical report, US-CERT, 2009.

[52] North American Electric Reliability Corporation. Reliability standards.

[53] North American Electric Reliability Corporation. Cyber security - electronic security perimeter. Technical Report CIP-005-4a, 2011.

[54] North American Electric Reliability Corporation. Cybersecurity - critical cyber asset identification. Technical Report CIP-002-4, 2011.

[55] Noah Shachtman. Exclusive: Computer virus hits u.s. drone fleet. *Danger Room*, Fri, 07 Oct 2011.

[56] Allen-Bradley. Df1 protocol and command set: Reference manual. Technical Report 1770-6.5.16, Allen-Bradley, 1996.

[57] Dimitrios Hristu-Varsakelis and William S. Levine. Handbook of networked and embedded control systems. 2005.

[58] Modicon. Modicon modbus protocol reference guide. Technical guide, MODICON, Inc., Industrial Automation Systems, 1996.

[59] Mael Horz. HxD - freeware hex editor and disk editor, 2011.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 22 Mar 2012	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From - To)* Aug 2010 - Mar 2012

4. TITLE AND SUBTITLE External Verification of SCADA System Embedded Controller Firmware	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER HSHQDC-11-X-00089
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) McMinn, Lucille, R. 2d Lt	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, Wright-Patterson AFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/12-02
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of Homeland Security ICS-CERT, POC: Eric Cornelius, DHS ICS-CERT Technical Lead, ATTN: NPPD/CS&C/NCSD/US-CERT, Mailstop: 0635, 245 Murray Lane, SW, Bldg 410, Washington, DC 20528, email: ics-cert@dhs.gov phone:1-877-776-7585	**10. SPONSOR/MONITOR'S ACRONYM(S)** DHS ICS-CERT
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Current supervisory control and data acquisition (SCADA) systems do not have sufficient tailored electronic security solutions. Programmable logic controllers (PLCs) are particularly vulnerable due to a lack of firmware auditing capabilities. Indeed, PLCs are field devices that directly connect to end physical systems for control and monitoring of operating parameters -- compromise of PLC firmware could have devastating consequences. This research presents a tool we developed specifically for the SCADA environment to verify PLC firmware. The tool does not require any modifications to the SCADA system and can be implemented on a variety of systems and platforms. The tool captures serial data during firmware uploads and then verifies against a known good firmware executable binary. The tool can also replay captured data and analyze firmware without the presence of a PLC. The ability to isolate the tool from production systems and adapt to various architectures, along with its portability, make the tool a viable application for SCADA incident response teams and security engineers.

15. SUBJECT TERMS

SCADA system cyber security, programmable logic controller, firmware verification, serial data capture, PLC emulation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Maj Jonathan W. Butts
U	U	U	UU	99	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x4332 jonathan.butts@afit.edu